USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613–5000

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

Functional Description and Formal Specification of a Generic Gateway

(ASQBG-C-89-020)

August 1988

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332–0800

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704--188 Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: NONE
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT: UNLIMITED
2b. DECLASSIFICATION / DOUWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S): ASQBG-C-89-020
6a. NAME OF PERFORMING ORGANIZATION: University of Arizona; 6b. OFFICE SYMBOL (if applicable):	7a. NAME OF MONITORING ORGANIZATION: AIRMICS
6c. ADDRESS (City, State, and ZIP Code): Computer Engineering Research Laboratory, Electrical and Computer Engineering Depart., University of Arizona, Tucson, Arizona 85721	7b. ADDRESS (City, State, and Zip Code): 115 O'Keefe Bldg., Georgia Institute of Technology, Atlanta, GA 30332-0800
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AIRMICS; 8b. OFFICE SYMBOL (if applicable): ASQBG - C	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:
8c. ADDRESS (City, State, and ZIP Code): 115 O'Keefe Bldg., Georgia Institute of Technology, Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62783A	DY10	00-08	

11. TITLE (Include Security Classification)

Functional Description and Formal Specification of a Generic Gateway (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)

ChangWon Son and Ralph Martinez

13a. TYPE OF REPORT	13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT (Year, Month, Day): 1988, August	15. PAGE COUNT: 225

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Generic Gateway, Internet Gateway, ISO Protocols, LANs, WANs, CCITT Standard Protocols, Subnetwork Communications

19. ABSTRACT (Continue on reverse if necessary and identity by block number)

This report is divided into three parts: 1) Overview of the Generic Gateway, describes the functional requirements of internet gateways, 2) Formal Specification of the Generic Gateway by LOTOS, uses an ISO formal protocol specification language to describe the functional operation of the generic gateway, and 3) Testing of the Generic Gateway with CLIPS, describes the proposed use of an AI constraint language to perform verifiable testing of the generic gateway specification.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED / UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Allan R. Osborn	22b. TELEPHONE (Include Area Code): (404) 894-3136	22c OFFICE SYMBOL: ASQBG - C

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

FINAL REPORT

Contract No. B-10-695-SI
U.S. Army Institute for Research in Management
Information, Communications, and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

FUNCTIONAL DESCRIPTION AND FORMAL SPECIFICATION OF A GENERIC GATEWAY

by

ChangWon Son
Research Associate

and

Ralph Martinez, PhD
Director

Computer Engineering Research Laboratory
Electrical and Computer Engineering Department

THE UNIVERSITY OF ARIZONA
Tucson, AZ 85721

August 1988

TABLE OF CONTENTS

TABLE OF CONTENTS -- continued

TABLE OF CONTENTS -- continued

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS - Continued

CHAPTER 1

INTRODUCTION

The generic gateway project sponsored by AIRMICS contributes to gateway design approach, formal specification of communication systems, and modeling and testing of communication systems with an AI based language. This section introduces the work performed to generic gateway task of the Interoperable Global Information System (IGIS) project.

For incompatible networks interconnection, various gateways have been designed in the last few years. Our work is different from the traditional gateway design approaches from several points of view. One of the significant differences is the independent design on each half gateway. For example, while the traditional gateway design approaches seek tight link between gateway halves, the generic gateway approach seeks independence between them. And the gateway halves are interconnected by a linking block which is called a Protocol Negotiation Block (PNB). The PNB resolves the mismatches between dissimilar subnetworks, and the process of mismatch resolution is called negotiation.

Secondly, the formal specification of the proposed generic gateway have been demonstrated by the formal specification tool, LOTOS (Formal description techniques based on the temporal ordering of observational behavior). There are a few reasons of favoring LOTOS for our generic gateway specification. First, LOTOS represents most communication system's characteristics fairly well. Secondly, LOTOS provides an abstracted description without being bothered by the detailed aspects of the system. However, LOTOS has some disadvantages. LOTOS is not in a mature state yet [DIS 8807]. Also a compiler for LOTOS is not available which can be used during test or verification of the specification. The disadvantages will disappear with time and development.

Finally the generic gateway will be demonstrated by verifiable testing with CLIPS. CLIPS was developed by the Artificial Intelligence Section of the Mission Planning and Analysis Division at NASA/Johnson Space Center. As a constraint oriented AI language, CLIPS provides us a nonprocedural execution environment which is suitable to analysis of multiprocessing and non-deterministic system behavior. As an AI tool, CLIPS allows us to demonstrate the intelligence features which are desirable in the generic gateway.

1.1 Background

During the past decade, computer networks have been evolved due to various demands and purposes. In most cases they are dissimilar. If they are used independently the dissimilarity is not a serious problem. Major corporations have a proliferation of Local Area Networks (LANs) and Wide Area Networks (WANs). These networks are mostly incompatible. During the last few years the necessity of internetworking has increased dramatically due to the following reasons:

a). Corporations are decentralized and dissimilar computer networks exist at each site.

b). Information contained on computers in the corporate networks must be shared across internetwork environment.

c). Lack of internetwork standard and incompatible networks have made interconnection of networks a goal for corporate users.

For the interconnection between incompatible networks, the following two approaches have been suggested: a) Protocol convergences which suggest to re-design existing network protocols to recently standardized protocols, such as International Standards Organization (ISO) protocols; b) There are protocol conversions, which convert or translate network protocols between incompatible networks.

While the convergence effort gains more popularity, a few arguments have been aroused by a number of experts [Green 86]. As reference claims, the convergence can not be a satisfactory solution for the following reasons:

a) Standardization is already late and various networks widely exist and they are quite incompatible each other to be converged.
b) Network technology is not in a mature status yet, so complete migration to a standard might cause to limit the new technology.
c) Different network requirements means that standardization can not specify all possible network application.
d) Incomplete standard specification for computer networks.
e) User favoration where the corporate user favors one network's characteristics over another.
f) Incomplete understanding of networking and internetworking problems.

Hence the incompatibility problem will be around for a while. The second approach called, protocol conversion, will be a necessity until convergence can be reached as a final stage.

Protocol conversion, which is also referred as a protocol translation or mapping, can be subclassified by direct conversion and indirect conversion methods. Direct conversion is one-to-one protocol mapping scheme between two interconnected network's protocol sets. On the other hand, an indirect conversion scheme requires conversion of each network's protocols to an intermediate protocol. This intermediate protocol is referred as a neutral protocol. Suppose network M and network N need to communicate with each other and they are incompatible, then one protocol conversion is required by the direct conversion, which converts protocols between network M and network N. But, the indirect conversion requires two protocol conversions, one on network M side from network M protocol to an intermediate network protocol, and another on network N side from network N protocol to intermediate network protocol. In many cases the protocol conversion is done by a protocol convertor, called internet gateway. The gateway is responsible for providing a logical connection between two dissimilar networks by converting their network protocols.

Suppose N non-compatible networks are interconnected, then direct conversion requires N(N-1)/2 types of protocol convertors. On the other hand indirect conversion only requires N types of protocol convertors one for each network. While the direct protocol conversions have been practiced widely, The indirect conversions have began to gain more attention recently. This is due to the fact that the direct conversion schemes require a large number of gateways in the growing internetworking environment. The availability of internationally acceptable network protocols such as X.25 (CCITT) help the indirect protocol conversion scheme, as intermediate networks.

Indirect conversion has following limitations:

a) Protocol Overhead - The protocols must be translated twice, first to neutral protocol and then to target protocol,

b) Limited convertibility - The portion of protocol which convertible between network A and network B, might not be convertible by the limitation of intermediate network protocol X.

As a solution to these limitations, an alternative solution is proposed here. This approach is referred to as a generic gateway approach. The generic gateway has two major characteristics. First, a subnetwork protocol is converted to a universal structure (predefined internetwork service definition). Secondly, any intercommunicating subnetwork's protocols, which are already converted into universal form, are negotiated to resolve the mismatches. In this strategy, each network only requires a single type of internetworking units(gateways) without experiencing significant protocol conversion overhead. More specific descriptions of this approach will be given in the following chapters.

Gateway implementation is always a problem issue. Whether the protocol conversion is direct or indirect, the traditional gateway implementation schemes follow the phases.

a) Analyze both network protocol specifications.
b) Find a compatible functions and services.
c) Find an incompatible, but convertible functions and services.
d) Implement the functions and services which are found in a) and b).

Our proposed generic gateway has some differences on design phases from above. These are:

a) Understand the general network services requirement which are independent to any specific network
b) Formalize the internetwork services based on a)
c) Design and implement the subnetwork independent part of the gateway
d) Analyzes particular subnetwork protocol which are going to be interfaced by the generic gateway
e) Implement the subnetwork dependent part of gateway

by steps a) to d).

As a conclusion to this section, we state that the generic gateway approach is defined as an indirect protocol conversion between two or more dissimilar subnetworks. However, their design approach is significantly diverse from the traditional gateway design approaches as stated above. By that, we mean that each type of subnetwork only requires to provide one type of protocol conversion scheme to support connectability to any other dissimilar networks.

1.2 Scope of Document

This document is structured into the following three parts:

a) Overview of the Generic Gateway - This part describes the functional requirements of internet gateways

b) Formal Specification of Generic Gateway by LOTOS - This part uses an ISO formal protocol specification language to describe the functional operation of the generic gateway

c) Testing of the Generic Gateway with CLIPS - This part describes the proposed use of an AI constraint language to perform verifiable testing of the generic gateway specification.

CHAPTER 2

INTERNET GATEWAY'S FUNCTIONALITY

Internet gateways, regardless of their application environment, must perform a variety of functions in order to provide communication between incompatible networks. These functions cover protocol processing, performance, and operational aspects of the gateway. Some of them are as follows [MAR 87A]:

2.1. Medium Transformation - A gateway must translate messages between different transmission media, such as LAN RF broadband or baseband digital signals, and the serial 1822 or X.25 interfaces of the DDN packet switching nodes. Signaling schemes to each network must be present in the gateway.

2.2. Media Access Translation - The media access schemes on the LAN side of the gateway must be present in the gateway. Media access schemes on LANs, such as CSMA/CD or token passing 802.4, must be present in the gateway. Access schemes to the DDN must also be present.

2.3. Address Translation - Network addressing schemes are different on each network, so that the gateway must perform address translation. For example, the IEEE 802.3 LAN uses a 48 bit flat addressing scheme and the DDN uses a 32 bit two-level addressing scheme. The gateway must recognize internet addressing schemes when interconnecting multiple networks.

2.4. Protocol Transformation - The network protocols of each network must be transformed through decapsulation and encapsulation steps in L part of the gateway. For the DDN, the Internet Protocol (IP) and the Transmission Control Protocol (TCP) must encapsulate to a LAN message. The LAN protocol headers must be stripped before hand-off to the TCP/IP protocols. In the case of internet environment, a gateway-to-gateway protocol must be implemented.

2.5. Message Buffering and Flow Control - The gateway must be able to buffer messages from each network and flow control the network interfaces when the buffers are full. The flow control mechanisms buffer sizes are critical to the performance of the gateway.

2.6. Reliable Connection Management - The gateway must provide an error free link between two end-users on the networks by adhering to the error control and re-transmission mechanisms in the network protocols. The status of the connection must be made available to the user when error conditions arise.

2.7. Fault Detection and Reporting - The gateway must be able to detect connection status when establishing and maintaining a connection between two end-users. The gateway then reports to the users the condition of the links, gateways, and networks in the connection path, if a problem should occur.

2.8. Performance Monitoring and Statistics - The gateway must be able to monitor its performance relative to packet throughput and network routing statistics. These parameters can be read locally or remotely from the gateway and used for internet management.

2.9. Security Control Mechanisms - The gateway must adhere to internet security control and management procedures. This might include generation and routing of encryption keys and cryptographic algorithms.

2.10. Real-Time Response - The gateways must process packet traffic from the networks in real-time so that user response times are not compromised. The gateway must accommodate the differences in network response times. Real-time response is also important during interactive user sessions. The gateway must sustain the communication rates of each network.

2.11. Parallel Processing Architecture - The gateway must contain parallel processing architecture to sustain the network transmission speeds. Dedicated protocols and communication modules must exist to achieve the performance throughput required by connection to multiple networks. This is an architecture implementation feature which aids in achieving real-time response.

2.12. ISDN Interfaces - Gateways must eventually interface to integrated services digital networks (ISDNs) for data, voice, and video communications. The gateway must interconnect to ISDNs and their predecessors.

2.13. Multiple Network Interconnection - Gateways must have the interfaces and link parts to access multiple communications systems and networks. Multiple networks connections through the gateway must be accommodated.

2.14. Multi-Level Security and Key Distribution- Communications between end-users in some internet systems will require multi-level security for sensitive information and the gateway must preserve the data security characteristics of several networks. The gateway must be able to adhere to these requirement.

2.15. Dynamic Network Topology Reconfiguration - Since the gateways are interconnected to multiple networks it is feasible to use the gateway to keep network status and give this information to the network management function for reconfiguration when nodes and networks fail or are destroyed.

2.16. Network Reachability - The gateway must determine the reachability and availability of neighboring networks. Network status must be exchanged between gateways so that alternate network hops be taken when a path is down.

2.17. Internet Management and Control - The gateway interacts with an Internet Management and Control Center to assist in the daily operation and reliability of the networks. The gateway performance monitoring function collect performance data and presents it to the Control Center.

CHAPTER 3

INFORMAL GENERIC GATEWAY SPECIFICATION

3.1 Generic Gateway Structure

The generic gateway structure is composed of three parts, two subnetwork dependent parts and a subnetwork independent part. The subnetwork dependent part is responsible for communication with its subnetwork nodes. The subnetwork independent part is a generic functional part which is responsible for the interconnection between two or more subnetwork dependent parts. The structure of generic gateway is illustrated in Figure 3.1 and discussed in this section.

The basic design strategies used for the generic gateway specification are as follows:

a) Divides the gateway functions into subnetwork dependent functions(protocol functions), and subnetwork independent functions (generic functions).

b) Subnetwork dependent part of gateway does not support any generic functions.

c) Protocol mismatches are solved only by the subnetwork independent part of the gateway.

d) Subnetwork independent part of gateway does not support subnetwork protocol functions.

e) The communications between the subnetwork dependent and independent part are done via service access points(SAPs).

The parts of the generic gateway are described next. The subnetwork dependent part is decomposed into a Subnetwork Medium Access Block(SMAB), and a Subnetwork Protocol Block(SNPB), The SMAB supports the physical and data link layers protocols of each subnetworks. The SNPB supports the Network and Transport layer protocol of each subnetwork. The subnetwork independent part is consists of a Protocol Negotiation Block(PNB) which binds the two subnetworks and a Data Base Block(DBB). The Data Base is a shared memory between subnetwork dependent parts of gateway and the network independent PNB. The DBB contains the following information:

a) subnetwork characteristics, such as the maximum allowable packet size, sequencing methods, timeout value.

b) subnetwork statistics such as packet load and throughput status.

c) The routing information which is used by PNB and the gateway-to-gateway protocol during connection establishment phase.

For the proper gateway operation, a third kind of module is required. This is the gateway-to-gateway protocol (GGP) module. Like the data communication protocol supporting blocks, the GGP module can be subdivided into a removable GGP(local network GGP, or LGGP) and a fixed GGP(internet GGP, or IGGP). The functionality of the GGP and the LGGP and IGGP will be described in section 3.2.3.

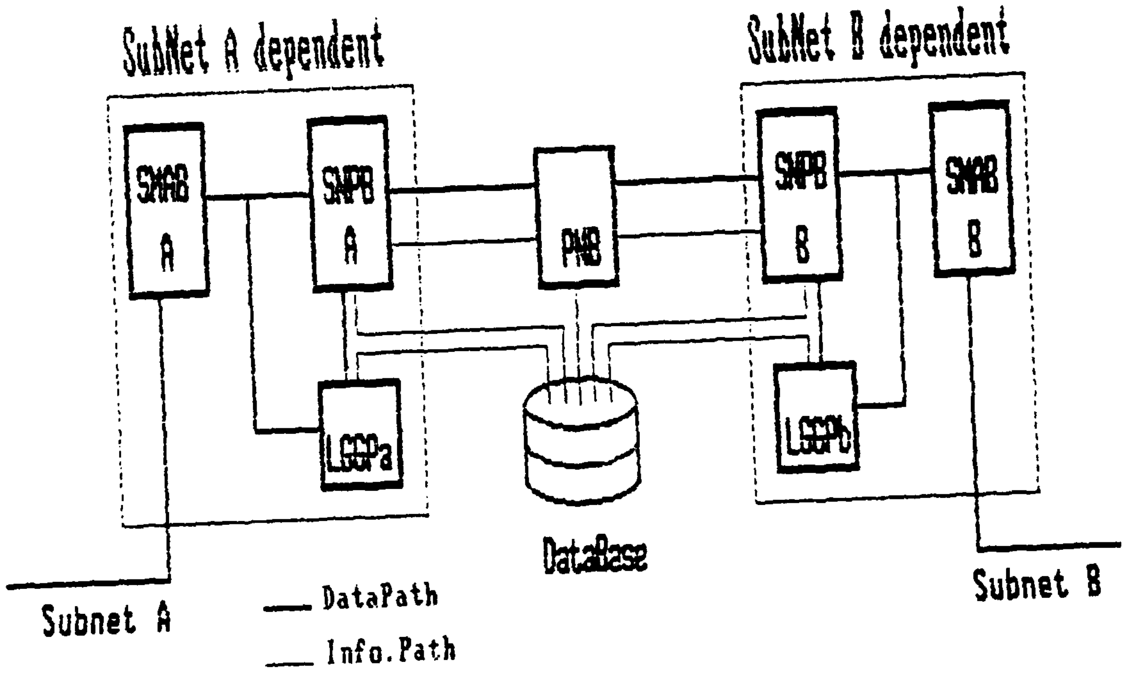


Figure 3.1 Generic Gateway Structure

3.2 Subnetwork Dependent Blocks

The subnetwork dependent blocks are the traditional part of gateway. These parts are responsible for communication between the subnetworks. Like most half gateway parts, the subnetwork dependent part supports the subnetwork protocols. However, the subnetwork independent part differs from the traditional gateway halves. For instance, the traditional gateway half converts the protocol structure and semantics to that of the other side gateway half. But in the generic gateway it converts them into an intermediate structure and semantics. The intermediate network structure and semantics are recognizable by the protocol independent block, called the PNB. The subnetworks dependent blocks are consist of two parts, a SMAB (subnetwork medium access block) and a SNPB (subnetwork protocol block). Details of the SMAB and SNPB are described in the next two sections.

3.2.1 Subnetwork Medium Access Block(SMAB)

As the name implies, the SMAB is responsible for the control mechanism to access the subnetwork transmission medium. The control mechanism includes the Physical and Data Link protocol layers. Because the characteristics of transmission medium and media access schemes are diverse for each individual subnetwork, more specific operational descriptions are out of scope of this document and will not be described in detail here. Data Link protocols include token passing (IEEE 802.4, 802.5) and CSMA/CD (IEEE 802.3). Physical layer signaling and transmission medium include coaxial broadband and baseband, TDMA, twisted pair, and fiber optics.

3.2.2 Subnetwork protocol block (SNPB)

The basic functions of the subnetwork protocols in the SNPBs follow the basic philosophy of the OSI reference model. However, the protocols might differ on syntax and semantics. As stated previously, the individual subnetwork's protocol characteristics are varied due to the specific subnetwork. In this section some important issues of the subnetwork protocols are discussed.

The first issue to be addressed on the gateway design is that of protocol conversion between Network layers. If the gateway provides too much protocol conversion, such as up to the Application layers, then the complexity of the gateway could be a performance bottleneck in the internet. If the gateway supports too few layers, then each node on the subnetworks must provide an excessive number of protocol services. The main issue here is whether the internet protocols are centralized or distributed. Centralization means lower performance and decentralization means complex individual nodes. For instance, the ISO specifications recommend the Network layer protocol for intermediate network nodes, and gateways[ISO 7498]. The DDN selects the Transport protocol, for its gateway protocol level[RFC-74]. Other gateways use Network layer routers and lower layer MAC bridges as their protocol conversion levels. In the generic gateway, the transport protocol layer is used for protocol conversion.

Another issue is the type of service subnetworks provide, connectionless or connection-oriented. Interconnection of connectionless and connection-oriented subnetwork requires extensive protocol conversion. The connection-oriented services must be emulated by the SNPB in the gateway. The advantages of connectionless and connection-oriented network approaches are in the flexibility and reliability respectively. Connectionless internetwork environment offers versatility, efficiency, flexible topologies, load sharing, and administrative manageability. The connection-oriented internetwork environment provides reliable data transmission, error recovery, congestion control, and reliable network management. Wide area networks (WANs) use connection-oriented approach because their error rates are generally higher than a LAN's and reliable service is mandatory.

Another important characteristic of subnetwork protocols is the quality of service on each subnetwork. Quality of service requires reliable data transport and is an important feature of subnetwork classification. In this case, reliability does not necessary mean the Transport protocol, even though it is confused in many occasions. Reliability implies error free data transmission between communicating nodes. In some cases, the quality of service implies message delay time versus cost, such as a priority or precedence of services.

From the above discussions, the subnetwork's characteristics can be summarized as the follows:

a) SNPB provides either Network or Transport level service.
b) SNPB provides either connectionless or connection-oriented service.
c) SNPB provides either reliable or unreliable data transport service.

Depending on the application, each subnetwork can provide more then one type of services on each category. For example, subnetwork A can provide connectionless and connection-oriented services, and multiplex between the two.

3.2.3 Local Gateway to Gateway Protocol Block (LGGP)

An internetwork environment may contain several networks and gateways. In this case, the gateways must collaborate to find the best path between network nodes. The gateways on the subnetwork require to exchange the information. This information includes gateway, link and node status. The information exchange is provided by a local gateway-to-gateway protocol(LGGP). The LGGP provides the following services and functions.

a) Address resolutions on reachable nodes.

b) Routing information exchange.

c) Name service.

d) Network statistic information exchange.

e) Universal time service.

Unlike an autonomous internetwork system the generic gateway does not expect that interconnected subnetworks to share a common gateway protocol. However, each subnetwork is responsible for providing a gateway protocol locally, thus the LGGP. For instance, if one of the subnetworks is a part of DARPA Internet, then a few gateway protocols may be involved. Examples of these are the Gateway-to-Gateway Protocol (GGP)[RFC 823], Exterior Gateway Protocol (EGP) [RFC 904], Routing Information Protocol (RIP) [COM-88]. The gateway protocols may be quite complicated and they are strongly subnetwork dependent. The internal gateway protocols of an internet will not be discussed in here. More information can be found in published documents and articles [COM-88].

3.3 Subnetwork Independent blocks

While subnetwork dependent blocks provide the most protocol functional support on each subnetwork, the subnetwork independent blocks provide a group of gateway specific functions. These functions include resolution of protocol mismatches, address resolution, and routing. Basically the subnetwork independent blocks are categorized by the following three functional blocks.

a) DBB (Data Base Block) - which maintains all internetwork information;

b) GGP(gateway to gateway protocol) - which is responsible to resolve the routing and addressing problems.

b) PNB (protocol negotiation block) - which resolves most protocol incompatibilities between dissimilar subnetworks.

These parts of the generic gateway are described in detail in sections 3.3.1 to 3.3.3 below.

The subnetwork independent blocks must meet the following requirements:

a) Independence of Subnetwork Protocol - The gateway must be able to operate with a wide variety of subnetwork characteristics. An open systems architecture in the subnetwork independent blocks is required.

b) Completeness - The gateway must provides enough functional support for the subnetwork functional protocols. This implies that the gateway must provide functionality which might be expected by the subnetworks.

c) Functional flexibility - The gateway must be flexible enough to compensate the differences between two interconnected subnetworks. When any specific functions are not provided by one of the subnetworks, the gateway must have a capability to resolve the mismatches. This is done either by providing extended functions on the subnetwork which lacks in the functions or by eliminating the functions during communication. The former method is preferred.

d) Knowledgeable - The gateway must have enough knowledge to make the required translation of protocol functions between the subnetwork protocols. The gateway must be well informed about each subnetwork's characteristics. This knowledge is kept in the data base portion of the gateway.

e) Extendability - The gateway must have a capability to extend the functional capability when required, without disturbing current gateway operation. This implies that the gateway must be designed with the incremental functions.

f) Negotiability - When the subnetwork's functions are mismatched and they cause the interoperability problems, they must be negotiated by the gateway to resolve the mismatches. This is performed by the Protocol Negotiation Block (PNB) in the gateway.

The generic gateway requirements discussed above are more idealistic than practical. To archive practicality, the requirements must be restricted or the internetwork environment must be limited to bounded network domains. The limitation on requirements for the generic gateway are as follows:

a) Subnetworks Limitation - The suite of subnetworks is limited to the networks which are structurally and functionally compatible, or softly incompatible networks. Examples are IEEE 802, X.25, ISO, SNA, TCP/IP, and MAP networks. The protocol matches for these networks are well understood.

b) Supported layers limitation - The gateway only provides a protocol conversion up to transport layer. The main reason is that by reducing the number of converted layers, the complexity of the gateway becomes feasible to implement. Unlike the Application layer protocols, the lower layer protocols are well defined and their varieties are reasonably small.

c) Functional Separation - The generic gateway must be decomposed, into two parts, subnetwork dependent part and subnetwork independent part. The subnetwork dependent part is responsible to handle all subnetwork protocol functions. The subnetwork independent part provides all protocol conversions. The quick overview of above three blocks will be discussed in the following sections.

3.3.1 Data Base

The data base maintains all gateway information which includes the subnetwork information and the information of individual connections, the connection status table. Also, it maintains other information which are required for the gateway maintenance.

3.3.2 Network Protocol Negotiation Block (PNB)

The PNB is responsible for linking the internetwork services between two subnetworks. Internetwork services, which are requested from one of the gateway's subnetworks, are converted or negotiated to the other subnetwork's service structure and semantics. The services are then transported to the second subnetwork.

The PNB provides the following negotiations:

a) Connection schemes negotiations.

b) Data transport negotiations.

c) Subnetwork parameter negotiations.

More specific negotiation phases will be introduced on the following sections.

3.3.3 Internet Gateway to Gateway Protocol Block (IGGP)

The gateways require to exchange the network information to utilize the internetwork resources. In the section 3.2.3 the intra-subnetwork gateway protocols (LGGPs) have been discussed. Unfortunately, in the generic gateway, the two (or more) interconnected subnetworks can not be pre-defined and the LGGPs can not be predicted to be interoperable. Hence, the processes which link the individual LGGPs are required. These processes are called the internet gateway to gateway protocol (IGGP). The major function of the IGGP is interconnection of services between subnetwork LGGPs via standard service access points. The services include address service, route service, and congestion control service. However, the detailed internal functions of the IGGP will not be introduced in this document, but only the results of the IGGP and LGGP are assumed be available during the connection establishment phase on the PNB. DDN's gateway protocols, EGP (Exterior Gateway Protocol) and GGP (GGP) of DDN can be considered as LGGPs.

3.4 Generic Gateway Services

Two groups of services can be expected from the generic gateway. They are: a) Protocol services which have been originated by the protocol itself, b) Gateway services which are dedicated to the gateway. The first group of services includes connection establishment and termination, and data transport services. The second group of services includes routing, protocol conversion, and protocol negotiations services. The first group of services are common on all network nodes while the second group of services are only available on the gateways (or intermediate nodes). The first group of services should be explicitly accessible by the network users, but the second group of services, in many cases, are transparent to most network users.

Because these two groups of services are strongly related each other, we have chosen not specify the generic gateway using service definition. Instead, the specification will be presented by the communication phases, such as a connection establishment, a data transport, and a connection termination phase.

3.4.1 Gateway Initialization

Before the gateway involves any internetwork services, the fundamental information of each subnetworks must be collected by the gateway and conformance of the subnetwork's interoperability should be made. The stage in which the gateway organizes the information and builds strategies is called as a gateway initialization phase. During this phase the gateway performs one of important negotiations, called static negotiation.

During static negotiation the invariant subnetwork information will be tested whether the subnetworks can communicate with each other or not. The result of the static negotiation could be one of three as follows: 1) Success, the two subnetworks may communicates each other without degrading any services, 2) Minor restriction, the two subnetworks may communicate each other, however some internet communication services are restricted, and 3) Failure, the interoperability of the two subnetworks is not attainable.

The following subnetwork information could be used in the static negotiation phase:

a) Subnetwork protocol type.

b) Subnetwork reliability.

c) Connection and packet delivery times.

3.4.2 Connection Establishment

The connection establishment phase is one of the most complicated part of the computer communication protocol. Several functions should be handled during this phase. The destination user address and network must be found. A proper path between source and destination or routing path is required. And the connection must be acknowledged by the communication end points or synchronized.

a) Connectionoriented and Connectionless subnetworks - In general, computer networks are characterized as connectionoriented and connectionless networks. The connection oriented network maintains a logical link which called as session, during which communications takes place. This phase of the communication during the link setup is called the connection establishment phase. On the other hand, the connectionless network transports individual messages without maintaining a logical link between them.

b) Initial Synchronization - One of the major functions of connection establishment phase is the synchronization between the source (calling) node and destination (called) node. This synchronization has two major purposes. First, it makes sure both end entities (nodes) are ready for the communication and, second, it negotiates on the communication parameters (or functions). For instance, the sequence number for the initial packets could be exchanged between communicating entities during the connection establishment phase. Logical and physical addresses of source and destination nodes may be also be exchanged.

c) Number of packets - The number of required packets during connection establishment phase classifies the connection establishment phase as handshaking schemes. More specific handshaking schemes are presented below.

In the one way handshaking scheme, which requires only one packet, the connection can be established by a simple procedure. The calling node sends a connection request (CR) packet to a called node. Then the calling node assumes that the connection is completed without verifying the connection status.

In the two way handshaking scheme, the connection phase requires two packets one for each direction. The calling node requests a connection by sending a CR packet to a called node. Then the called node sends the connection confirm (CC) packet back to the calling node. The calling node waits until the CC is received from the called node before proceeding any further actions, such as an initiation of data packet transmission. Because the two way handshaking requires more packets, it is more complicated and slow. However, it has a few important advantages. First, of all the unnecessary data packets transmission can be eliminated when the destination node is not currently available. More importantly it can provide a flexible communication environment. One of the well known example is negotiations on classes of services between transport layer protocols in ISO modeled networks.

The two way synchronization does not guarantee that the communicating nodes are synchronized during connection establishment phase. Lost CC packets can not be easily detected by the called node, and the called node could enter the data transfer phase without knowing the calling node's status. To achieve synchronization, a third packet is required which tells whether the calling node actually receives the CC packet or not. This scheme is called three way handshaking. The DoD networks are well known example of this scheme. More information on three handshaking is contained in referred in [RFC 793].

While subnetworks have quite diverse connection management schemes, their interconnection by a generic gateway requires special care. The following strategies are proposed in the generic gateway:

a. The subnetwork dependent part of gateway (subnetwork protocol supporting block) follows the strategy which have been employed in the local subnetwork.

b. However, the communication between subnetwork dependent part(SNPB) and independent part(PNB) must be predefined in universal (connection establishment) strategy.

c. In our generic gateway the PNB expects two-way handshaking scheme if the subnetwork is connection-oriented network.

A few scenarios of the connection establishment phases are as the follows:

NOTE: The following acronyms are used in the scenarios:

CR:	Connection Request
CC:	Connection Confirm
DR:	Data Request
SYN:	Synchronization
ACK:	Acknowledgement

Scenario 1:

Connection-oriented to Connection-oriented connection
(2 way - 2 way sync.)

```
(Source)    SNPBa          PNB        SNPBb    (Destination)
=============================================================
CR  -->
            CR --------->
                           CR -------->
                                       CR ------>
                                             <------- CC
                             <--------- CC
                                      * Connection Established
                   <-------  CC
                           * Connection Established
   <-------  CC
             * Connection Established
```

Note: In Scenario 1, the connection between two connection-oriented subnetworks is established by CR and CC. As noticed here the connection establishment status is propagated from the destination to the source node. As a result, the destination node can send data packets immediately after it sends the CC. On a positive side, the expedited packets can be transferred before CC reaches the source node. On the other hand, this might waste resources by transmitting the data packets through a path which might no longer exist.

Scenario 2:

Connection-oriented to Connection-oriented connection
(2 way - 3 way sync.)

```
(Source)    SNPBa         PNB          SNPBb      (Destination)
============================================================
  CR ------>
               CR -------->
                           CR -------->
                                       CR -------->

                                          <---- ACK,SYN

                              <------- CC
                                       ACK -------->
                                      * Connection Established
               <---------- CC
                                 * Connection Established
    <---------- CC
               * Connection Established
```

Note: In Scenario 2, the destination (called) subnetwork requires synchronization using the 3 way handshaking strategy, while the source (calling) subnetwork provides a two-way handshaking strategy. By that, the packet exchange must be adjusted by the gateway. One interesting point is that the acknowledge will be returned from the SNPB on the destination side of the gateway, not by the final (calling) node. Thus, it will produces the same result as the first scenario, (connection is established from destination side nodes first).

Scenario 3:

Connection-oriented to Connectionless connection (2 way)

```
(Source)    SNPBa          PNB         SNPBb     (Destination)
============================================================
 CR ---->
            CR -------->
                          DR ------->
                                      DR   ------>
                                                <------ ACK
                             <------- ACK
                    <-------- CC
                                  * Connection Established
   <-------- CC
            * Connection Established
```

Note: In Scenario 3, the connection-oriented subnetwork initiates a CR to the connectionless subnetwork. The CR will be sent as a data packet, which has an empty data field, unless the source contains some data to be sent. Then the destination node will send the ACK back to the source node. Finally the source node receives the CC and the connection is established.

The main argument here is how sequential data packets can be delivered properly. One solution is, transfer each packet independently without any concern about the connection path which have been made during connection establishment (each sequential packet is treated as a connectionless data packet). The other solution is, transfer the sequential packets through the connection which is established during the connection establishment phase.

If the data transmission is in half duplex mode (only one node is allowed to send a stream of packets especially the one which initiates the connection), then the first scheme is sufficient. However if the connection must be maintained in full duplex mode (communication is allowed in both directions), the first scheme will confront a serious problem. THe problem is that each data unit transmission initiated by a connectionless subnetwork requires a connection establishment and its termination on the connection oriented subnetwork.

As a proposed solution to this problem, the connectionless subnetwork side of the gateway also requires maintaining a connection reference table which could be referenced during the data units transport with connection oriented subnetwork. The connection reference table would be removed when the disconnection request (DR) has been issued by the connection-oriented subnetwork or when a time out is reached on the connectionless subnetwork.

Scenario 4:

Connectionless to Connection-oriented connection
(2 way)

```
(Source)    SNPBa          PNB         SNPBb      (Destination)
===============================================================
 DR ----->
            DR -------->
                          CR ------->
                                       CR ------>
                                            <------- CC
                            <------- CC
                                      * Connection
                                        Established
                 <------   ACK
                       *Connection Established
  <------   ACK
```

Note: Scenario 4 consist of a connection establishment phase between connection-oriented and connectionless subnetworks, and it is the counter direction of Scenario 3. For communication from a connectionless to a connection-oriented subnetwork, there are two possible scenarios. The first treats each data packet as an independent entity, so that each packet initiates a connection request, complete data transfer, and terminates the connection. Because it is simple, it become one of the most popular methods between connectionless and connection-oriented network interconnection. However, it will involve too much overhead by the frequent connection establishments and terminations during data transports. As an alternative solution, which is suggested in the Scenario 3, the connection table method is proposed here. The first data packet from the connectionless network can be treated as a connection request packet. Then the subsequential packets will be transferred using the connection table. However, the main problem in this scheme is how and when the connection should be closed. The simplest solution is to allow sufficient amount of idle (inactive) connection between communicating parties in order to terminate the connection. Alternately, the connection can be terminated by the disconnection request from the node on the connection-oriented subnetwork.

c. Protocol Negotiation

The protocol negotiation functions are one of the most significant aspects of the generic gateway. The various connection establishment schemes between interconnected subnetworks noted above can be expressed as a part of the protocol negotiation functions. In this section other negotiations which can be performed by the generic gateway will be discussed.

In general the incompatibility of the dissimilar networks falls into one of three categories: 1) all incompatible protocol functions can be converted; 2) most of incompatible protocol functions can not be converted; 3) a few of the incompatible protocol functions can not be converted but the others can.

In most cases the incompatibility between networks fall into class 3, so that the most protocol functions are commonly found from both networks with minor syntactic or semantical differences (soft mismatch). However, some of the functions would not be on both networks (hard mismatch). The first type of incompatibility can be solved by the gateway, but the second type of incompatibility can not. Therefore, whether the protocol conversion is possible or not, mainly depends on the protocol functions which are not common in the both subnetworks. This also implies that the possibility of the protocol conversion (or gateway) depends on the willingness of giving up of the inconvertible functions. In the generic gateway, the willingness is expressed as a protocol negotiation.

In our current design the parameter incompatibilities have been chosen as major negotiation examples. Because they are fairly easy to approach and identify. Examples of these parameters will be given bellow. The parameter negotiation has two phases: 1) during connection request packet(CR) transfer; 2) during connection confirm(CC) packet transfer. Suppose a subnetwork A requests a connection to a subnetwork B by sending a connection request packet. The first negotiation is taken place when the gateway receives a connection request from the calling node, with the information of each subnetwork's pre-stored parameters and the parameters which are received as a part of the CR packet. The negotiation is performed by the Protocol Negotiation Block (PNB). If the first part of negotiation fails the connection is terminated by informing the connection denial reasons. Otherwise, the connection process can be continued. The second negotiation is performed when the gateway receives the connection confirm packet from the called node. In both cases, the PNB has two options: take the negotiated values or reject the values, and disconnect the connection.

Example:

Connection-oriented to Connection-oriented connection
(2 way - 2 way sync.)

```
  (Source)    SNPBa        PNB        SNPBb   (Destination)
============================================================
 CR ----->
              CR ------->
                        * first negotiation phase
                          CR -------->
                                      CR ------>
                                          <------- CC
                             <------- CC
                                    * Connection Established
                         * second negotiation phase
                 <------- CC
             * Connection Established
    <------- CC
*Connection Established
```

A few of interested negotiated parameters are as listed bellow.

Time Related Parameters Negotiations:

- Max. allowed delay on connection establishment,
- Max. allowed lifetime of a packet,
- Max. allowed delay on subnetworks,
- Max. allowed turn around time,
- Max. allowed inactive connection time.

Quality of Service Negotiations:

- The trade off between allowed error rate and throughput.

Packet Size Negotiations:

- Max. allowed Packet size,
- Blocking,
- Segmentation.

Other Functional Negotiations:

- Buffer size,
- Packet sequence number,
- Data transmission handshaking.

3.4.3 Data Transfer

After connection is established between communicating nodes, the data units can be transferred. This stage of connection is called the data transfer phase. If the data transfer is done in an ideal environment (error free environment) then the gateway's data transfer phase might be quite simple. It receives a data unit from one subnetwork then it sends the data unit to the other subnetwork. Unfortunately, the real communication environment is far from the ideal condition and a lot of unexpected events can happen, such as lost or duplicated packets. For these reasons, a number of protocol functions are dedicated to provide the reliable data transmission.

The reliable subnetwork must provide the following conditions: 1) the data units created by the data unit originator (source node) must be delivered to the final (destination) node; 2) the data units must be delivered in order and only once. To satisfy the requirement, complex communication mechanisms are provided in most network protocols, which detect and recover any errors during data units transfer such as lost, duplicated, or out of sequenced data units.

One of the crucial questions which has been asked is how reliable should the network be. Everybody understands that a reliable network provides better satisfaction than an unreliable network. However, higher reliability is not free. Reliability and throughput performance are contentional properties in general. Reliable communication requires complex processes such as retransmission schemes for lost or corrupted data transmission. So, the answer of the question must be sought by balancing between performance and reliability.

The reliability of a network, which is referred to as a quality of service in our gateway design, is a negotiable property during connection establishment phase. When real communication is concerned, the negotiation of reliability is quite complicated due to the various sources affecting reliability. In this section, we defined a structure to the notion of reliability for negotiation purposes. We define two levels of reliability on each category as stated bellow:

a) The subnetwork can be either reliable network or unreliable network.
b) The subnetwork provides either reliable data transport protocol or unreliable data transport protocol (virtual circuits of datagram service).
c) The user expects reliable data transport service or unreliable data transport service.

The quality of service negotiation can be built as a set of production rules. For example, the calling user requests a connection with reliable data transmission then the negotiation can be represented as bellow:

```
negotiation rule1:
 If (    (MyService is reliable)
      and
           ( (MySubnetwork_Protocol is reliable) or,
                (MySunbetwork_Quality is reliable))
         and
              ( (DestSubnetwork_Quality is reliable or,
                (DestSubnetwork_Protocol is reliable))
        )
   then
        Negotiated_Quality is Reliable_transmission;
```

As long as the protocol negotiation on the quality of service is done successfully during the connection establishment phase, the nodes may communicate with each other on the negotiated quality of service class.

Suppose current application requires reliable data transmission service, the interconnected networks are not reliable, and the network protocols provides reliable data stream service. Then the two interconnected subnetworks can provide a reliable data transport service by the reliable data transport control mechanisms with the protocols on each subnetwork.

```
 If (    (MyService is reliable)
      and
          ( (MySubnetwork_Protocol is reliable) or,
            (MySunbetwork_Quality isnot reliable))
      and
            ( (DestSubnetwork_Quality isnot reliable or,
            (DestSubnetwork_Protocol is reliable))
     )
 then
     Negotiated_Quality is Reliable_transmission;
```

But, unfortunately the mechanisms on reliable communication between subnetworks may not compatible each other, for instance one subnetwork uses TCP while the other uses TP-4. The question aroused here is how the two reliability control mechanisms can be interoperable with each other. When the interconnected subnetworks are pre-defined, the conversion between the reliability control mechanism may converted with each other. Unfortunately, this assumption may not available in the generic gateway approach. Thus, we propose the policies as follows:

a) On each side of subnetwork the reliable data stream control is maintained locally, which is independent to the other side of subnetwork.

b) The only reliable data packets are expected from the other side of subnetwork. Such as the other side of network is guaranteed or assumed that a stream of data packets, delivered, is error free.

The summary of this section is as the follows:

a) The minimum requirement for the data transport service will be selected by the quality of service negotiation during connection establishment phase. If the negotiation is completed successfully then data communication can be continued. Otherwise, the connection will be rejected.

b) The subnetwork dependent block (SNPB) provides a reliable communication with its local network's nodes. The packets will be sent to the subnetwork independent block (PNB). At this moment the packets are assumed error free or the service is not a reliable data stream.

a) Scenario 1

* Subnetwork A and B are acceptable by application

```
============================================================
   User A     SNPBa          PNB         SNPBb        User B

   Issue Packet
              -->
                             -->
                                           --->
                                                     Accept
                                                     Packet
```

Note: There are two cases, first the network service does not require reliable data transport, second the subnetwork A and B are reliable enough.

b) Scenario 2

* Subnetwork A is reliable but Subnetwork B is not reliable and the Subnetwork can support reliable data transport protocol

```
============================================================
    User A     SNPBa        PNB         SNPBb        User B
   Issue Packet
              -->
                            -->
                                     * Controlled
                                       By protocol X
                                       (ie. TP-4)
                                         ---->  *
                                                Controlled
                                                    By
                                                Protocol X
                                                Accept Packet
```

Note: The reliable data transport is maintained by the network protocol on the subnetwork B.

c) Scenario 3

* Reverse direction of case b)

```
===========================================================
   User A     SNPBa       PNB          SNPBb        User B

   * Controlled
     By Protocol X
            (ie. TP-4)
     Issue Packet
          --->
            * Controlled
           By Protocol X
               (ie. TP-4)
             --->
                            --->
                                         -->
                                                    Accept
                                                    Packet
```

Note: The reliable data transport is maintained by the network protocol on the subnetwork A.

d) Scenario 4

* Both subnetworks relies on the protocol and they are not compatible

```
============================================================
   User A     SNPBa       PNB          SNPBb        User B

   * Controlled
     By Protocol X
        (ie. TP-4)
     Issue Packet
           --->
            * Controlled
           By Protocol X
                (ie. TP-4)
                          --->       * Controlled
                                       By Protocol Y
                                        (ie. TCP)
                                          -->  *
                                               Controlled
                                                   By
                                               Protocol Y
                                               Accept
                                               Packet
```

Note: The subnetwork A and B maintain the reliable data transport. They are not compatible each other. By that, the control mechanisms are maintained independently.

e) Scenario 5

* Both subnetworks relies on the protocol and they are compatible

```
============================================================
   User A      SNPBa         PNB          SNPBb         User B

   * Controlled
     By Protocol X
        (ie. TP-4)
     Issue Packet
           --->
                             --->
                                            -->     *
                                                      Controlled
                                                          By
                                                      Protocol X
                                                      Accept
                                                      Packet
```

Note: The reliable data transport protocol bypasses gateway. Because the two end nodes use the same protocol for the reliable data communication. For an example if the communicating subnetworks are OSI networks then the reliable data transport can be maintained only by the end nodes.

3.4.4 Connection Termination

Releasing a connection between network users is a relatively simple task compared to connection establishment or data communication. The connection release can be distinguished as a user termination and a network termination. The first termination is evoked by a user when user finishes the internetwork communication session. The second termination is evoked by the network, or more specifically by the gateway. The first termination (disconnection) request by a network user, is a normal communication procedure. However, the second termination is generated by a gateway when it confronts any error conditions such as negotiation failures, improper transmission medium, excessive retransmissions, and network congestion.

The generic gateway distinguishes the two types of connection termination cases. First, if the termination request is originated by a user as a normal procedure, the gateway closes the connection in a graceful manner. Any undelivered packets must be delivered before it closes the connection. Otherwise the connection must be terminated immediately.

CHAPTER 4

FORMAL SPECIFICATIONS

4.1. Introduction

Formal specifications of communication programs begin to appear as early as 1970s. The formal specification languages have been widely studied during the last few years. In the survey of Kalnin'sh [KAL 86], 60 formal specification and verification techniques(languages) were identified. Each formal specification techniques has its own strong and weak points. However, the individual comparison of the specifications will not be discussed in this paper. Further studies have been done by [KAL 86], [SUN 81], [SCH 81], and by many other researchers. This section deserves concepts from the literature on specification and verification techniques, and uses them to describe the functionality of the generic gateway. First, we review the characteristics of formal specification.

The difficulty of formal specification of communication system is due to the following underling reasons.

a) Distributed System - The communication programs or systems are physically distributed, which results in the information not available in shared memory, but shareable by communication.

b) Non-deterministic Operation - The communication program can not be formalized as a deterministic system.

c) Concurrent System Behavior - The communication system requires multiprocessing environment, and a concurrent operating system must be provided.

d) Time Bounded - Due to the safety control (ie. deadlock recovery, lost packet detection etc.) each state is time bounded.

e) Event Driven - The state transition is event driven. Packet arrival or expired timer causes major state transitions in the communication systems.

4.2 Background of Formal Specification

In short, a specification is a detailed functional, performance, and operational description of a system (in our case it is a description of a gateway). Verification is a proof of correctness of the specification. Specifications provides a formal description of a system. They provide a high level of abstracted model of the system and thus avoids implementation details. Specifications provide a mathematical or logical foundation (algorithm) for verification, modeling and simulation, and implementation.

After the specification is done, correctness of the specification could be proven by the verification, before any system's implementation exist. The refinement can be done on the specification level with the result of verification.

As Kalnin'sh [KAL 86] stated that there are three level of specifications such as the follows: first, abstract specification (level-1) which describes overall structure of a system; second, design specification (level-2) which provides a complete description of functions of the system; and implementation specification (level-3) which provides the most detailed specification of the system. In DoD military standard terminology, these correspond to Type A, B, and C specification.

Level-1 specification is the first stage of specification which hides all details of a system. This stage of specification contains only a "What should it do" description without describing "How to do." The service specifications of the OSI protocols are a few examples of Level-1 specification, or Type A level specification.

Level-2 specification explains the more depth of the system which includes, "How" the system should operate to provide the services, which have been specified by the Level-1 specification. For example we can consider the protocol specifications of the OSI as Level-2 specifications. These are Type B level specification.

Finally Level-3 specification is an implementation level specification which will guides the system's implementation. It describes how the actual system should be generated. In many cases the Level-3 specifications are strongly bonded with the implementation languages, such as programming languages. These are Type C level specifications.

These specifications are required to describe the functional, performance, and operational aspects of a system.

4.3. Survey of Existing Specification Schemes

4.3.1 Finite State Machines

One of the oldest and the most popular models of specifying computer program's and communication systems is finite state automata(machines) techniques. Finite state automata (FSA) machines include a tuple of a state variable, a set of transition functions, a set of input/output variables, and a initial state. The finite state machines can be represented as graphical forms such as flow charts, or tabular forms such as state transition tables. The major drawbacks of these schemes are: 1) the states of the systems must be globally visible by single state variable; and 2) the number of states must be finite and must be kept reasonably small. In most communication systems the number of states could be nondeterministic and they might be unbounded, such as unbounded queues. A few improvements have been made such as abstract machines which allows more than one state variable. However, there are other restrictions such as poor capability in data representations (v-expressions) even though they give a good expression for the control expressions (p-expressions).

4.3.2 Petri Nets

Petri Nets are one of the common representation tools for the specification of communication systems. Petri Nets represent most common communication system's characteristics, such as concurrent operation, synchronous communication, and non deterministic control. Petri Nets can easily represent the infinite states conditions such as an unbounded buffer. A disadvantage of Petri Nets is their graphical representation natures, which limits the number of nodes specified.

4.3.3 Formal Languages

A relatively new class of specifications have recently emerged. They are mathematical representations in the form of algebraic specifications and formal programming languages. The algebraic specifications, such as abstract data type and axiomatic specification, provide a mathematical base for verification, especially semi automatic verification. Several protocol specifications and verifications have been demonstrated with these schemes. The formal programming language approaches have a few advantages. For example, these methods allow the communication system's designer and implementor to use general software engineering techniques, and the usual program verification techniques. In general, the algebraic specifications tend to be more abstract and ambiguous then the formal languages. They require high mathematical understanding from users. On the other hand, the formal programming languages are more readable and less abstract.

4.3.4 ISO Standards

The Formal Description Technique (FDT) groups of the ISO community have proposed standard specification languages for their formal description of OSI models. They are LOTOS (Language of Temporal Ordering Specification) [DIS 8807] and ESTELLE (Formal Description Technique based on an Extended State Transition Model) [DIS 9074] . These will be discussed in the sections below.

ESTELLE is more favored in the practical world, because it is based on traditional programming languages, such as Pascal. It is easily understand by the most programmers. The similarity between the specification language and the implementation language provides easy transition from specification into implementation.

On the other hand the LOTOS has been favored in academical world. LOTOS is based on the concepts of calculus of communication systems and abstract data type. Specification with LOTOS can be verified mathematically and provide more abstracted specification than ESTELLE. LOTOS specification have two components. The first component deals with the description of process behavior and interaction. The second component deals with the description of the data structure and value expressions. We have selected LOTOS to specify the generic gateway. It provides the concurrency features for multiple protocol processing and leads itself to an AI-based testing method.

4.4 Introduction to LOTOS

In this section, a few characteristics of LOTOS will be presented to aid the understanding of the formal specification of generic gateway. The more detailed LOTOS is contained in the OSI document [DIS 8807].

4.4.1 Abstract Data Type

One definition of abstraction is, "the process of forming a concept or idea by imaginatively isolating or considering apart the common characteristics of a group of distinct object or event." Subroutines and data structures are well known abstracted events and objects in computer programs. The term "abstracted data type" is referred to a class of object defined by a representation independent specification [GUT 75]. In general the abstract data type can be represented as a tuple of:

Abstract_Data_Type ::= <{operator}, {carrier}, {equations}>.

In LOTOS an abstract data type is defined by names of data carriers and operations. The names of data carriers are referred to as sorts. The sorts and operations of a data type are referred to as the signature of the data type. It defines the syntax of a data type. Below is an example of a type definition of the natural numbers. The definition has the name 'Nat_numbers', so that it may be distinguished from other data types. The signature of 'Nat_numbers' consists of the single sort 'nat', and the operations '0' and 'succ'. 'succ' can be applied to a single element of sort 'nat' and yields also an element of 'nat' as its result. '0' is an operation that has no argument, and therefore does not depend on any value. Such operations are called constants.

```
Type  Nat_numbers is
sorts nat
    opns 0:    -> nat
    succ: nat -> nat
endtype
```

Terms represent elements of a sort. If the denoted element belongs to sort s it is said that a term is of sort s, and is referred as the s-term. For example nat-terms are:

    0, succ(0), succ(succ(0)), ...

Each of these nat-terms can be interpreted as one element of the algebra of the natural numbers. In the Nat_numbers above we can add a new operator '+':

```
opns '+':  nat, nat -> nat.
```

With the new operator '+' new terms may be produced, i.e. succ(0) + succ(succ(0)). This construct is an equation. With the introduction of the equation and terms with variables, we now have the tools to specify property of operation. For example, a correct definition of the '+' operator is given by:

```
eqns forall x,y:nat
  ofsort nat
       x + 0 = x;
       x + succ(y) = succ(x + y).
```

The first equation expresses the behavior of the operator when it combined with the constant '0'. The addition with non-zero number is given by the second equation.

4.4.2 Behavior Expression

In LOTOS, distributed systems are described in terms of processes. A system as a whole is a single process that may consist of several interacting subprocesses. These subprocesses may in tern be refined into subprocesses. Therefore, a specification of a system in LOTOS is essentially a hierarchy of process definitions.

In LOTOS, each process can be imagined as a black box, or object, that is capable of communicating with its environment. The mechanisms inside of this box are not observable. The process communicates with its environment by means of interactions. The atomic form of interaction is an event. An event is a unit of synchronized communication that may exist between two processes that can both perform that event.

The communication element consists of a label, which is referred to as gate, and a finite list of attributes. Two type of attributes are possible: a value expression,and a variable expression. The value expression describes a data value and is preceded by an exclamation mark. A variable expression has a form '?x:t', where x is a variable name and t is sort identifier of the variable. The sort identifier indicates the domain of values over which x ranges.

In LOTOS, the communication method is different from traditional communication descriptions, such as a sender and receiver relationship. Instead, the interacting parties offer value or variables through the synchronized gates. Examples of communication via synchronized gates are given here.

Examples:

where: P1, P2: processes

g, a: gates

P, Q: behavior expressions

x: any variable (message).

a) P1: g!x and P2: g!2 then

P1 accepts '2' as its x value and the processes P1 and P2 continue their own processes.

Note: The x is a variable which means that process P1 will accept any value which is offered from any other party.

b) P1: g!1 and P2: g!3 then

the two processes offer different values, both process will wait until any other process accepts those values.

Note: In this, case if another process P3 offers gate!x then any one of the process P3 will accepts any one of the value, 1 or 3, and those agreed pair will continue the process. Choice of the process may be non-deterministic.

c) P1: g?x:int and P2: g!3 then

P1 accepts the value 3 as its value of x and both process will continue process.

Note: In this case the ? has the special meaning that P1 is willing to accept any value.

4.4.3 Basic Operators in LOTOS

Nondeterministic (Choice)

P [] Q - The only one of the process P and Q will be chosen and the choice is non-deterministic.

Parallelism

P |[a]| Q - The process P and Q will progress simultaneously, but they have to be synchronized by the activities on the gate 'a'.

P ||| Q - The process P and Q will be in progress simultaneously but they do not need to be synchronized.

Disable

P [> Q - Q can be in active state only before or during the process of the P process no later than that.

Enable

P >> Q - Q can be in active state only after the process P terminates the execution successfully.

Guarded process

[condition] -> Q - Q can be executes only if the propositional condition is satisfied.

Hiding

hide g in P - The gate 'a' is a local gate so it can not be observed from the outside of the process P.

Internal event

i{P} - The process P is internal events which means that the operation of the process can not be visible but only result can be exported.

Behavior Identifier

process [gates] (parameters) - It is the same kind of procedure specification in the usual programming languages.

4.5 Verification Techniques

While the formal specification techniques are used to define communication systems, verification techniques are used to insure their correctness. Therefore, the verification could be used during the design phase before any system implementation exists, in order to avoid possible design errors. In general, the verification schemes can be formally classified as "reachability analysis" or "program proving" methods.

4.5.1 Reachability Analysis

Reachability analysis techniques exhaustively explore all the possible interactions of two (or more) entities within a system. From a given initial state, all possible transactions(user commands, time-outs, message arrivals) are generated, leading to a number of new global states. This process is repeated for each of the newly generated states until no new states are generated. In this way all possible system states are reached and tested.

Reachability analysis is well suited to check the general correctness properties of specifications. These properties are direct consequence of the structure of the reachability graph. Global states with no exits are either deadlocks or desired termination states. Similarly, situations where the processing for a receivable message is not defined, or where the transmission medium capacity is exceeded are easily detected. The generation of the global state space for transmission models is easily automated, and several computer aided systems for this purpose have been developed.

The major difficulty of this technique is "state space explosion" because the size of the global state space may grow rapidly with the number and complexity of protocol entities involved and the underlying system's service. This verification is only efficient when it is used with limited break points. As noted, the reachability analysis methods are frequently used to verify the specifications which are based on the FSA.

4.5.2 Program Proving

The program proving approach involves the usual formulation of assertions which reflect the desired correctness properties. Ideally, these would be supplied by the service specification. But, as noted above, services have not been rigorously defined in most protocol work, so the verifier must formulate appropriate assertions of his own. The basic task is then to show (prove) that the protocol programs for each entity satisfy the high-level assertions (which usually involve both entities). Often this requires formulation of additional assertions at appropriate places in the programs.

A major strength of this approach is its ability to deal with the full range of protocol properties to be verified, rather than only general properties. Ideally any property for which an appropriate assertion can be formulated can be verified, but formulation and proof often require a great deal of ingenuity. Only modest progress has been made to date in the automation of this progress.

4.6 Requirements for Communication Systems Verification

Whether the reachability analysis or program proving is used, the following properties should be verified.

a) Deadlock Freeness - Deadlock is one of the most common problem for the communication systems. There are numerous causes of the deadlock in the communications systems. Some of them are preventable deadlocks (allocating limited resources). Some of them are recoverable deadlocks (the lost or corrupted packets). Then there are deadlocks by the incorrect specification.

b) Liveness - While the deadlocks lead to a "dead" state by the lack of a condition which leads the next state in the state transition, the liveness problem deals with the case when the next state is omitted from the specification (incomplete specification). A liveness proof must verify that from each reachable state any other state is reachable, or for each reachable state and event there exists a reachable state from which this event can occur.

c) Tempo-blocking Freeness(livelock freeness) - Unlike deadlock, the livelock is experienced while the processes are in live state. However, they can experience non-productive(non-progressive) infinite looping. For example, the unbounded lost and retransmissions of packets can cause livelock. The non-productive infinite loop freeness which is also called tempo-blocking freeness must be proved.

d) Starvation-freeness - When several processes contend for resources which become available infinitely many times, no process should be prevented forever from acquiring the resources that it needs. This property is also called fairness.

e) Recovery from failures - After a failure the protocol should return to normal execution state within a finite number of steps.

f) Self Synchronization - From any abnormal state, the protocol should return to a normal state within a finite number of steps (This property and recovery are closely related).

g) Correct Execution - The system specification must produces the same result which is intended by the designer.

4.7 Verifiable Testing

As Loeckx et al. [LOE 87] noted, "Program testing... may increase confidence in correctness of a program but it is far from guarantying that the program is free from semantic errors." Program testing might not be sufficient enough to prove the total correctness of the program. Formal verifications are the only way to prove the specification's correctness. Several verification techniques have been proposed and demonstrated during last decades [SCH 81a] [SHA 83]. Unfortunately, the verification techniques are not well established yet. In the most cases, the only simple systems have been verified and the complex systems verification experienced a great deal of difficulty.

The major difference between formal verification and testing is in the degree of abstraction of the system. First, the testing methods abstract the systems on the functional level and then the abstracted systems are executed with predefined events. On other hand the verification methods abstract the system not only on the functional level but also on the event level. When the system is abstracted only on the functional level, the system can be verified on the given set of events. In many occasions, it is referred as an experimental frame. On contrary, the verification may produces a proof for all possible events.

In this document the concept of verifiable testing is proposed. Verifiable testing (VT) is a testing method that differs from traditional testing methods (i.e., simulations) in the following aspects:

a) VT concentrates in the semantic or analytic property of system while simulation concentrates in the functional or quantitative properties.

b) VT tests verification properties on the given experimental events, such as liveness and safeness. Simulation only centers on the systems behavior for the experimental events.

Concrete verifiable test scheme is introduced in the Chapter 6 of this document. The verifiable testing schemes are based on the use of an AI-based constraint language. More specifics on verifiable testing are contained in Chapter 6.

CHAPTER 5

GENERIC GATEWAY SPECIFICATION WITH LOTOS

The generic gateway specification is based on the discussions from the Chapter 3 of this document. The generic gateway will be referred to as a gateway for the convenience. In this specification, the subnetwork dependent modules (SMABs) are not specified for the following reasons: a) Variety of the subnetworks - the diversity of the subnetwork medium access schemes are too great to be specified by one specification; b) The lower layers of protocols are strongly hardware related protocols; c) The lower layer protocols are not critical factors on this gateway formalization. As the result, the SNPBs are assumed as the lowest layer of the subnetwork protocols. The GGP is not specifically specified in the current specification. The GGP will be introduced in the next phase of the research.

The following specification policies are applied as the gateway specification environment:

a. Two types of subnetworks, connectionless, and connection-oriented subnetworks.

b. Cross network reliable data stream controls are not allowed. Each subnetwork requires to manage the reliable data transfer scheme individually (locally).

c. The gateway requires to provide multiple connectability.

d. Each communication session may be limited with in one connection and no more than one session may share the same connection.

5.1 Gateway System Structure

In LOTOS, the gateway is represented by a group of data types and a process. The data types are definitions of the objects which are accessed in the process. The data types are as messages or packets, connection reference and its list(table), and the subnetwork and gateway information. The gateway process consists of a number of subprocesses which individually represents each represented sub modules of the gateway, such as SNPB, PNB, etc. Each subprocess consists of its subprocesses and so on. Hence, the gateway specification is provided as a hierarchical structure. In this specification, the two types of subnetworks are defined, 1) connectionless, and 2) connection-oriented subnetwork. The high level structure of the gateway specification is:

```
Specification Generic_Gateway
    { global type definitions }
```

```
behavior Gen_Gateway

    process Generic_Gateway
        ....

    where

        (* connectionless subnetwork part *)
        { connectionless subnetwork local data types }
       process SNPB
               {connectionless subnetwork behavior }
         endproc

       (* connection-oriented subnetwork part *)
       { connection-oriented subnetwork local data types }
       process SNPB
               {connection-oriented subnetwork behavior }
      endproc

      (* protocol negotiation part *)
        { PNB local data types }
      process PNB
           { PNB behavior specification }
      endproc

   endproc
end specification.
```

The interactions between subprocesses to external environment are called as events. Each event consists of an event label and a finite list of event attributes. The event label, which is referred to as a gate, is a communication channel name by the traditional terminology. The attributes are messages on the channels. However, the gates and channels are functionally different from several points and there is no fundamental difference between sender and receiver. More detailed gate description is contained in the ISO LOTOS specification document [ISO 8807].

The gates can be either external interface gates or internal gates. If the activities on the gates are interaction with the external environment then the gates are referred to as interface gates. If the interactions are between subprocesses or with in a process then the gates are referred to as internal gates. The only difference between internal gates and interface gates is that the internal gates might be hidden from the external environment while the external gates may represent the behavior of the system by the activities on the gates. The gate 'a' and 'b' are internal gates, and 'pa' and 'pb' are internal gates, in the gateway structure as in Figure 5.1.

```
                        CORE
           +-------------------------------------------+
           |                                           |
           | a +--------+ pa +---------+ pb +--------+ b |
SMAB ------+ SNPB       +-----+ PNB      +-----+ SNPB     +-------- SMAB
 |         |   |        |    |         |    |        |  |         |
 |         |   +---+----+    +----+----+    +---+----+  |         |
 |         |                                           |         |
neta       +-------------------------------------------+       netb

                 gates:  a, pa, pb, b
```

Figure 5.1 Generic Gateway Communication Gates

5.2 Data Type Specification

As noted previously, the data types serve two roles. First, it specifies the message object definitions, such as communication packets. Second, it defines the network information such as the data base entries and the connection management information. Similar to the process structure, the data types are represented in a hierarchical structure. Complicated data type structures can be constructed by the relations of the simple data types. For example, the connection reference entries on each connection table can be defined from the simpler data type definitions such as 'address definition', 'queue definition', etc.

5.3 Intermodule Relation

In general, each service primitive or packet is decomposed into various fields. Each field is owned by a peer layer protocol and the use of the field strictly belongs to the peer layer protocol. For instance, when a transport layer sends a data unit to a network layer it attaches transport layer protocol information on the data unit packet (encapsulation). In the destination node, the reverse process is performed. The transport layer removes the transport layer specific information from the data unit (decapsulation) before it sends the data unit to the destination node upper layer. An example of this is shown in Figure 5.2. The SMAB on subnetwork A sends a service primitive to SNPB A, after it extracts the header of SMAB (Info1). Then the SNPB extracts its header (Info2) and it sends the service primitive to PNB. On the subnetwork B, new headers are attached (Info1' and Info2'). However the header structures may be different on both side subnetwork.

To identify the connection, each SNPB maintains a connection reference entry (SConRef) for each logical link between communication entities. The SConRefs on both SNPBs are linked by PConRef on the PNB. A service primitive on subnetwork A is referenced by SConRef and it is referenced by SConRef' on the subnetwork B. By separating the connection reference, each subnetwork module can maintain the connection status independently. For example, the packet sequence number can be individually.

```
         (A)                        SMAB                 (B)
+------+------+----------+              +---------+------+------+
|Info1 |Info2 | Data     |              | Data    |Info2'|Info1'|
+------+------+----------+              +---------+------+------+
   |       |        |                       |        |       |
   |      \|/      \|/          SNPB       \|/      \|/      |
   |     +------+----------+            +---------+------+   |
   |     |Info2 | Data     |            | Data    |Info2'|   |
   |     +------+----------+            +---------+------+   |
   |        |        |                      |        |       |
  \|/      \|/       |                      |       \|/      |
+---------------+    |                      |    +---------------+
| SConRef       |    |                      |    | SConRef'      |
+---------------+    |                      |    +---------------+
       |             |      +---------+     |           |
       |             +--> Data       <--+               |
       |                    +---------+                 |
       |                                                |
       |                  +-----+------+                |
       +------------------|     |      |----------------+
                          +-----+------+
                             PConRef
```

Figure 5.2 Inter Module Relation

5.4 SNPB Structure

The main body of SNPB which is named as SNPBStart consists with two processes, SNPBInit and SPNB are shown in Figure 5.3. During system initialization phase, the SNPBs will be initialized and the result of the initialization will be informed to the PNB by the process named 'SNPBInit'. Suppose the initialization step is not successful, then no further operation may possible. Only successful initialization will yield the gateway to further steps. If the SNPBs are successfully initialized and their internal information is transferred to PNB then the SNPBs can continue the main process 'SNPB' (see Figure 5.3).

```
process SNPBStart[a,b] :noexit :=
   SNPBInit[b](|info, cons)
 >>
   accept info:SNetInfo, cons:SConList in
   SNPB[a,b](info,cons)
where

  process SNPBInit[b] :exit (SNetInfo, SConList):=
    i(|Mynetinfo, cons)  (* internal function, which produces
                            subnetwork information and initial
                            connection reference list.     *)
  >>
    b ! Mynetinfo; exit (Mynetinfo, cons)
 endproc  (* SNPBInit *)
....
```

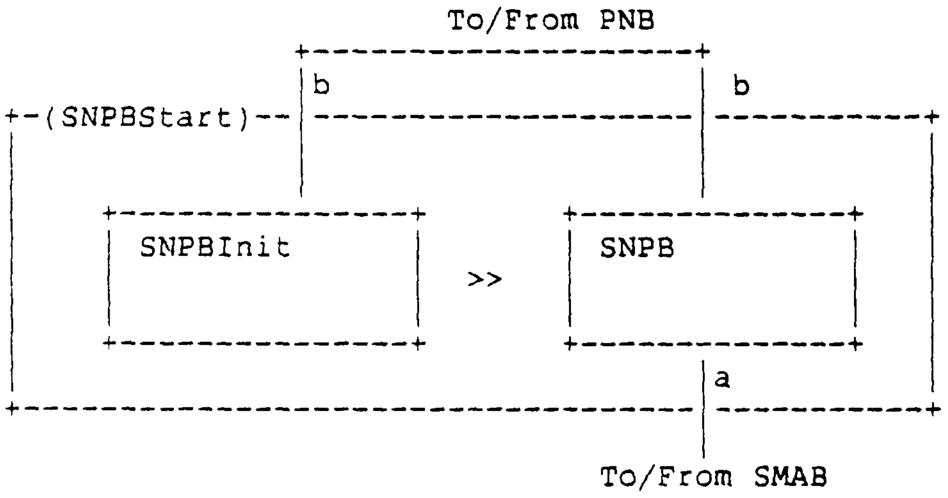


Figure 5.3 Structure of SNPBStart process

The process SNPB consists of four subprocesses which are two interface processes, a connection handler process, and a protocol process. Figure 5.4 shows their relationship in the SNPB. The subprocesses interact with each other with activities on the gates (internal gates). The four subprocesses are (see Figure 5.4):

a) SNPBUInterface - interfaces the protocol service primitives between SNPBProtocol and PNB,

b) SNPBDInterface - interfaces the protocol service primitives between SNPBProtocol and SMAB,

c) SNPBProtocol - provides subnetwork protocol services,

d) SNPBConHandler - manages connection reference list and links the protocol primitives to the appropriate connection reference.

```
process SNPB[a,b] (info:SNetInfo, cons:SConList) :noexit :=
    hide ga, gb, ds in

    (  SNPBConHandler [ds] (info, cons)
           |[ds]|
            (      SNPBDInterface[a,ga,ds] (info)
               |[ga]|
                   SNPBProtocol[ga,gb,ds] (info)
               |[gb]|
                   SNPBUInterface[b,gb,ds] (info)
            )
        )

 where
     process SNPBConHandler ... endproc
            process SNPBUInterface ... endproc
     process SNPBDInterface ... endproc
     process SNPBProtocol ... endproc

endproc (* SNPB *)
```

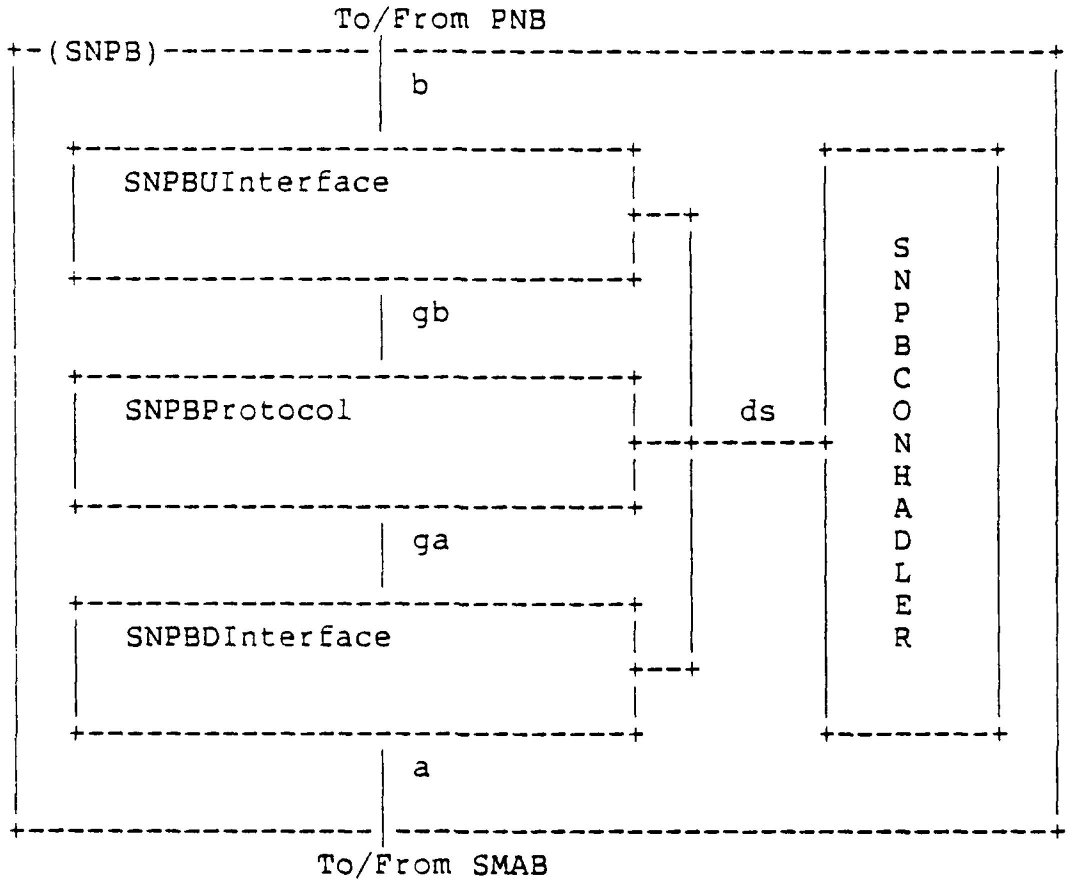


Figure 5.4 SNPB Structure and Subprocesses

5.5 PNB Structure

The PNB structure is similar to the SNPB structure above. Only, the inside of its behaviors are different. The some structure in Figure 5.4 can be applied for the PNB. The subprocesses structure is the same, ut the functions are different.

5.6 Peer Layer Service Relations

In general, the services of an n-layer are the capabilities which the n-layer offers to n-user (n+1 layer). The services are specified by describing the service primitives and parameters which characterize each service. A service may have one or more related primitives which constitute the interface activity which is related to the particular service. Each service primitive may have zero or more parameters which convey the information required to provide the service. Primitives are of the three general types [ANSI 802.2]:

REQUEST: The request primitive is passed from the n-user to the n-layer to request that a service be initiated.

INDICATION: The indication primitive is passed from the n-layer to the n-user to indicate an internal n-layer event which is significant to the n-user. This event may be logically related to a remote service request, or may be caused by an event internal to the n-layer.

CONFIRM: The confirm primitive is passed from the n-layer to the n-user to convey the results of one or more associated previous service request(s). This primitive may indicate either failure to comply or some level of compliance.

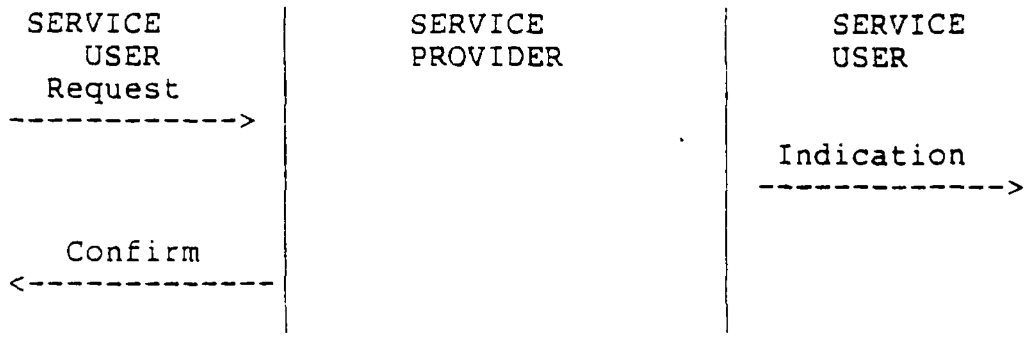


Figure 5.5 Service Primitives

In the OSI basic reference model, each peer layer of protocols are related each other. The N layer protocol is an user of N-1 layer protocol service and also it is N layer protocol service provider for the N+1 layer protocol. The peer layer's relationship can be applied to the generic gateway. The SMAB provides services for the SNPB and the SNPB provides services for the PNB. Figure 5.6 shows the generic gateway structure and its services between layers. However, the PNB service can not be defined as an service user and service provider relationship (PNB is not a peer protocol). Hence, the PNB services can not be defined in the peer protocol hierarchy because the services provided by the PNB are not protocol services.

```
      +---------------------------------------------------------+
      |                          PNB                            |
      +---------------------------------------------------------+
           |         /|\   SNPB Service User  |         /|\
-------------------------------------------------------------
         \|/          |      SNPB Protocol   \|/         |
      +---------------------+              +---------------------+
      |        SNPB         |              |        SNPB         |
      +---------------------+              +---------------------+
           |         /|\   SMAB Service User  |         /|\
-------------------------------------------------------------
         \|/          |      SMAB Protocol   \|/         |
      +---------------------+              +---------------------+
      |        SMAB         |              |        SMAB         |
      +---------------------+              +---------------------+
```

Figure 5.6 Gateway Services Structure

5.7 Connection Oriented SNPB Protocol Specification

5.7.1 Service Primitives

This section specifies the service required of the SNPB by the PNB as viewed from the PNB. This service allows a local SNPB entity to exchange packets with the remote SNPB.

5.7.1.1 Connection Establishment

The service primitives associated with connection establishment are:

Connection Request (ConReq),

Connection Indication (ConInd),

Connection Response (ConRes),

Connection Confirm (ConCnf).

The ConReq primitive is passed to the SNPB to request that a logical link connection be established between a local SNPBSAP (SNPB service access point) and a remote SNPBSAP. The ConInd primitive is passed from the SNPB to indicate the results of an attempt by a remote entity to establish a connection to a local SNPBAP. The ConRes primitive is passed to the SNPB to respond that a logical connection be established between the local SNPBSAP and a remote SNPBSAP. The ConCnf primitive is passed from the SNPB to convey the results of the associated ConReq primitive.

5.7.1.1.1 Connection Request (ConReqP)

Function: This primitive is the service request primitive for the connection establishment service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from PNB to the SNPB when the PNB wishes to enable a logical link connection, of a given service class to a local SNPB.

Effect on Receipt: The receipt of this primitive by the SNPB causes to the local SNPB entity to initiate the establishment of a connection with the local user node.

Additional Comments: The ConReqP primitive may initiate a new connection reference when the ConRef is not known by the PNB.

5.7.1.1.2 Connection Indication (ConIndP)

Function: This primitive is the service indication primitive for the connection establishment service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from SNPB to the PNB to indicate that a connection of a certain service class has been established.

Effect on Receipt: The PNB may use this connection for Internet work connection establishment.

Additional Comments:

5.7.1.1.3 Connection Response (ConResP)

Function: This primitive is the service response primitive for the connection establishment service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from the PNB to the SNPB to indicate that a connection of a certain service class has been established on the remote SNPB.

Effect on Receipt: The SNPB may use this connection for data unit transfer.

Additional Comments: The ConResP primitive indicate that the connection is established between the remote node and the remote SNPB.

5.7.1.1.4 Connection Confirm (ConCnfP)

Function: This primitive is the service confirm primitive for the connection establishment service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed to the PNB from the SNPB to indicate that a connection of a certain service class has been established on SNPB.

Effect on Receipt: The PNB may use this connection for Internetwork connection establishment.

Additional Comments: At this moment the connection is established between SNPB and local node. And the data units may be accepted.

5.7.1.2 Connection Release

The service primitives associated with connection release are:

Disconnection Request (DisReq),

Disconnection Indication (DisInd).

The DisReq primitive is passed to the SNPB to request that a logical link connection be released between a local SNPBSAP (SNPB service access point) and a remote SNPBSAP. The DisInd primitive is passed from the SNPB to indicate the results of an attempt by a local entity to terminate a connection to a remote SNPBAP.

5.7.1.2.1 Disconnection Request (DisReqP)

Function: This primitive is the service request primitive for the connection termination service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from PNB to the SNPB when the PNB wishes to terminate a connection.

Effect on Receipt: The receipt of this primitive by the SNPB causes to the local SNPB entity to terminate a connection.

Additional Comments: Receiving DisReqP primitive indicates that the connection on the remote side is not active any more by that further incoming data units from local subnetwork may be discarded.

5.7.1.2.2 Disconnection Indication (DisIndP)

Function: This primitive is the service indication primitive for the connection termination service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from SNPB to the PNB to indicate that the connection is terminated on local SNPB.

Effect on Receipt: The PNB may use this primitive for Internetwork connection termination.

Additional Comments: The local connection on the SNPB no longer exist by that no further data units will be accepted.

5.7.1.3 Data Transfer

The service primitives associated with data transport are:

Data Unit Request (DataReqP),

Data Unit Indication (DateIndP),

Acknowledgement Request (AckReqP),

Acknowledgement Indication (AckIndP).

By DataReqP primitives the data units will be received from PNB, and by DataIndP primitives the data units will be sent to the PNB. In either case the data units must be acknowledged.

5.7.1.3.1 Data Request (DataReqP)

Function: This primitive is the service request primitive for the data unit transfer service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from PNB to the SNPB when the PNB wishes to send a data unit.

Effect on Receipt: The receipt of this primitive causes the SNPB to send the data unit to the local node.

Additional Comments: The data which is received as a DataReqP primitive may be stored in the queue.

5.7.1.3.2 Data Indication (dataIndP)

Function: This primitive is the service request primitive for the data unit transfer service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from SNPB to the PNB to indicate the arrival of an data unit from the specified local entity.

Effect on Receipt: The PNB may sends the data unit to remote SNPB.

Additional Comments:

5.7.1.3.3 Acknowledgement Request (AckReqP)

Function: This primitive is the service request primitive for the data unit transfer acknowledgement service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from PNB to the SNPB when the previous data unit was transferred successfully to remote side.

Effect on Receipt: The receipt of this primitive causes the SNPB to attempt to send the data unit acknowledgement to local subnetwork node.

Additional Comments: The receipt of this primitive means that the previous data unit transfer was successful by that the data unit may be removed from the queue on the SNPB.

5.7.1.3.4 Acknowledgement Indication (AckIndP)

Function: This primitive is the service request primitive for the data unit transfer acknowledgement service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from SNPB to the PNB to indicate the arrival of an data unit from the remote SNPB.

Effect on Receipt: The PNB may sends the data unit acknowledgement primitive to remote SNPB.

Additional Comments:

5.7.2 Connection-oriented SNPB Connection Reference Structure

In general, gateways provide more than one connections or sessions simultaneously. Each service primitive and protocol function must be identified by the connection reference entity (ConRef). The ConRef may contain following information:

Addresspair: Source and destination address,

ConInfo: Various information, such as: subnetwork information, sequential number of data unit to send next, size of buffer for data unit queues, state of connection reference, etc.

Ques: Window queues for data transport.

5.7.3 Connection-oriented SNPB Elements of Procedure

5.7.3.1 General

This section specifies the individual connection entities on the connection oriented SNPB. The connection entity which is referenced by ConRef has following state phases: connecting, data transfer, and disconnecting. From state transition maps (Figure 5.7, 5.10, 5.11) the state 's1' to 'S7' are connecting states, and 's8' to 's10' are data transfer states. Finally, 's11' and 's12' are disconnecting states.

5.7.3.2 Connection Establishment Phase

The internetwork communication link can be initiated by a local subnetwork user or remote subnetwork user as shown in Figure 5.7.

a) By local subnetwork user:

The local user's connection request is received by

SNPB as a connection indication service primitive (ConIndM) from SMAB. The SNPB initializes a connection reference table for the specific connection entity. Then, it sends a connection request to the PNB as a connection indication service primitive (ConIndP). The connection request service primitive is sent to the remote user under PNB's and remote SNPB's control. The result may returned from the PNB to the local SNPB, either disconnection request(DisReqP) or connection confirm(ConCnfP). When the result of the connection is not successful (DisReqP), the currently created connection reference is released. No further interaction may allowed on this connection reference. If the result of the connection is successful (ConCnfP) then the SNPB sends the connection confirm to SMAB and the SNPB enters data communication phase.

b) By remote subnetwork user:

The remote user's connection request is received by SNPB as a connection request service primitive (ConReqP) from PNB. The SNPB initializes a connection reference table for the specific connection entity. Then, it sends a connection request to the SMAB as a connection request service primitive (ConReqM). The connection indication service primitive is sent to the local user under local SNPB control and the result is returned from SMAB to local SNPB either disconnection indication(DisIndM) or connection response (ConResM). When the result of the connection is not successful (DisIndM), the currently created connection reference is released. No further interaction is allowed on this connection reference. If the result of the connection is successful (ConResM), the SNPB sends the connection response to SMAB. Then the SNPB enters data communication phase.

When two independent connection establishments are initiated from both ends of the same pair of communication entities the connection establishment processes may collide with each other at the gateway. In this case, there are two options for the gateway: it rejects one of the connection request and it allows only one connection between the communicating pair; or it allows both connections between the communicating pair. In this specification, the first scheme is selected because it is more simple, and in some subnetworks multiple connection may not be allowed between any pair of communicating nodes.

In this specification, the two way hand shaking scheme is used for the communication establishment method. In this scheme, the called user may sends the series of data units before the calling SNPB sends a connection conform primitive to calling user. In this case the incoming data units may be stored in the queue.

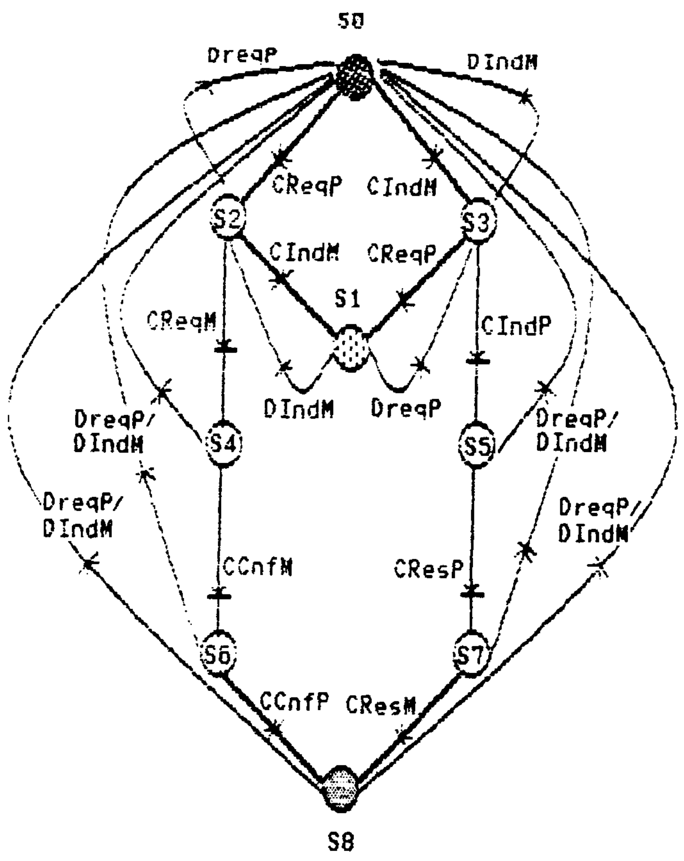


Figure 5.7 State Transition of Connection Establishment Phase

5.7.3.3 Data Transport Phase

In this specification, the data units are managed by two individual data queues one for each directional data transport. More specifically, the windowed queues are used in this specification. In the windowed, queue the queue element has locality information, viz. sequential number, and it may be inserted in any order but it can only be extracted in sequential order. For example, Figure 5.8, the new data packet Y which has been sequentially labeled x can by inserted at any moment. However, the packet can be removable when all preceding packets are inserted and also they have been removed.

```
(insert Y at x) ->+   (x: position)
        +----------v-------------------------------------+
        |          Y                                     |--
->       +-----------------------------------------------+
         ... x+1 x x-1 ....                t+1  t .... (remove)
```

Figure 5.8 Windowed Queue

a) Local node to remote node data transfer - One data unit transfer involves four service primitives on the SNPB. The complete sequence of operation is called a data transport cycle. The data transport cycle starts by receiving a data unit which created from local node by local SMAB and it is transferred to the local SNPB as a DataIndM. The date unit is transferred to PNB as a DataIndP. Then, the data unit is acknowledged by PNB as AckReqP. The data transport cycle is completed by sending the AckReqM to the local SMAB and finally to the local node. In this specification, the window scheme is used in the SNPB. The SNPB may receive a series of data units without being delayed by waiting for the acknowledgement from the remote node. The data units may or may not be received in order and are stored in the queue. The data units will be transferred to the PNB when the data unit become the expected data unit. In fact, the window queue is more complicated than provided in this specification. The recovery from the lost data units must be provided. But in this specification they are not be considered, because it involves complicated problems.

```
  DataIndP  AckReqP            AckIndP DataReqP
      |      \|/                  |      \|/
     /|\      |      Gate b      /|\      |
+-----|-------|-------------------|-------|---------+
|     |       |                   |       |         |
|   +-------------+             +-------------+     |
|   |             |             |             |     |
|   +-------------+             +-------------+     |
|   |             |             |             |     |
|   +-------------+             +-------------+     |
|   |             |             |             |     |
|   +-------------+             +-------------+     |
|   |             |             |             |     |
|   |   LocQue    |             |   RemQue    |     |
|   +-------------+             +-------------+     |
|     |       |                   |       |         |
+-----|-------|-------------------|-------|---------+
      |      \|/     Gate a       |      \|/
     /|\      |                  /|\      |
  DataIndM  AckReqM            AckIndM  DataReqM
```

Figure 5.9 Data Transfer Queues

b) Remote node to local node data transfer - This case is similar to above example, however the data units are transferred from the node on the other subnetwork. In a similar way, the data transport cycle involves four service primitives on the SNPB. The data unit is transferred from the PNB to the local SNPB as a DataReqP primitive. Then, the data unit is stored in the remote queue. It is sent to the local SMAB if it is the expected data unit from the local SMAB. Otherwise, it is stored in the queue.

There are two different strategies to send the acknowledgement back to the PNB. First, the SNPB sends the acknowledgement back as soon as the data unit is sent to the local SMAB. Second, it holds the acknowledgement until the data unit has been transferred correctly to the final node and the acknowledgement is returned from the final node of the data unit. The two strategies have strong and weak points of their own. First, if the acknowledgement was returned from the final node then the gateway may assure that the previous data units have been transmitted to the final node. That may cause a problem, especially when the each interconnected subnetwork has different transmission time such as a LAN to WAN interconnection. Second, if the acknowledgement is returned by the SNPB or SMAB the transmission time related problem is solved however the gateway may not see the complete data transmission status.

The state transition map of the SNPB data transfer phase is illustrated in Figure 5.10.

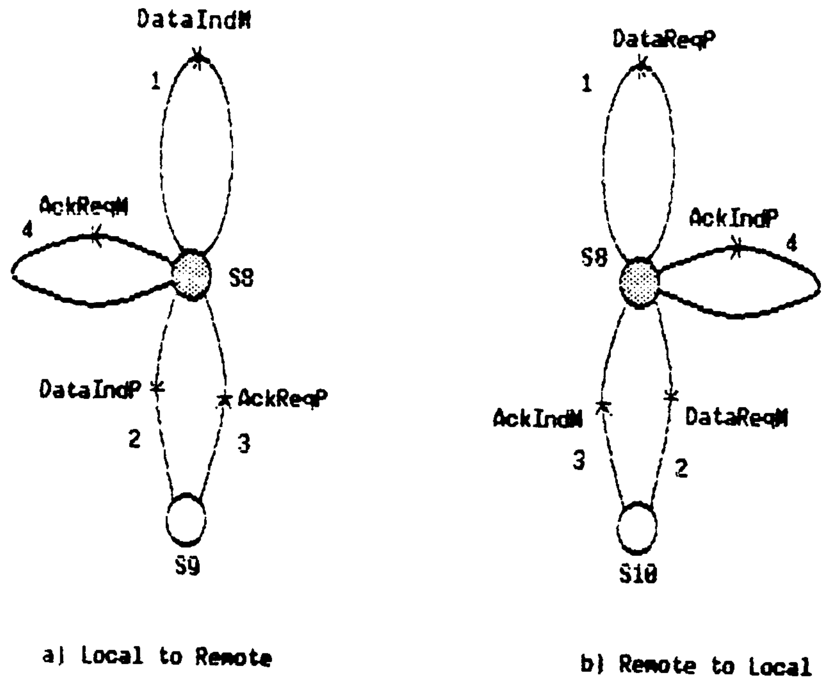


Figure 5.10 State Transition of Data Transfer Phase

5.7.3.4 Connection Termination

The connection termination requires one of three conditions, incomplete connection attempt, user request, and unrecoverable internetwork communication failure. The first and third conditions force the communication session to terminate immediately. The second condition allows the user to terminate the connection gracefully by delivering any non-transferred data units to the destination node.
The state transition map is illustrated in Figure 5.11.

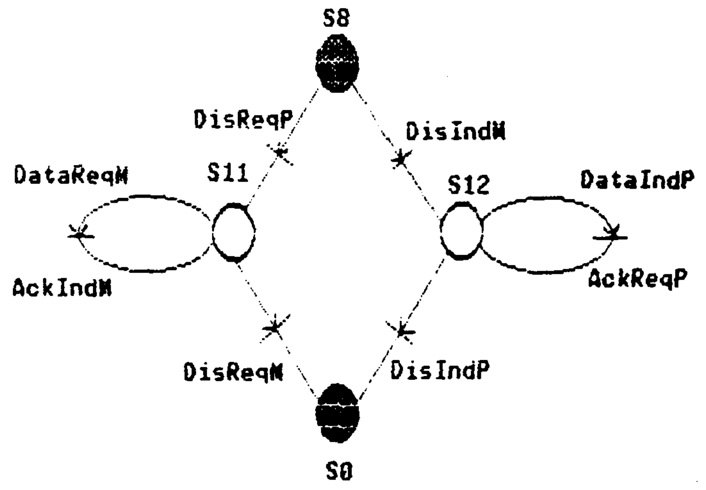


Figure 5.11 State Transition of Connection Termination Phase

5.8 Connectionless SNPB Specification

5.8.1 Service Primitives

The connectionless SNPB protocol service primitives are similar to the connection-oriented SNPB protocol service primitives. The data transport primitives are discussed in the following sections.

5.8.1.1 Data Request (DataReqP)

Function: This primitive is the service request primitive for the data unit transfer service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from PNB to the SNPB when the PNB wishes to send a data unit.

Effect on Receipt: The receipt of this primitive causes the SNPB to attempt to send the data unit.

Additional Comments: When the ConRef is not an assigned reference, viz. the data unit is the first data unit between communicating entities, then the ConRef will be created and the further communication between the communicating pair nodes will be referenced by the ConRef.

5.8.1.2 Data Indication (DataIndP)

Function: This primitive is the service indication primitive for the data unit transfer service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from SNPB to the PNB to indicate the arrival of an data unit from the specified local entity.

Effect on Receipt: The PNB may sends the data unit to remote SNPB.

Additional Comments: The first data units for the communication pair forces to create a connection reference(ConRef) which can be used during data transport between the communication node pair.

5.8.1.3 Acknowledgement Request (AckReqP)

Function: This primitive is the service request primitive for the data unit transfer acknowledgement service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from PNB to the

SNPB when the previous data unit was transferred successfully to remote side.

Effect on Receipt: The receipt of this primitive causes the SNPB to attempt to send the data unit acknowledgement to local subnetwork node.

Additional Comments: Receipt of this primitive means that the previous data unit transfer was successful by that the data unit may be removed for the queue on the SNPB.

5.8.1.4 Acknowledgement Indication (AckIndP)

Function: This primitive is the service indication primitive for the data unit transfer acknowledgement service.

Semantics of the Service Primitives: The primitive shall provide parameters as follows:

ConRef,

PNPkt.

When Generated: This primitive is passed from SNPB to the PNB to indicate the arrival of an data unit acknowledgement from the specified local entity.

Effect on Receipt: The PNB may sends the data unit acknowledgement primitive to remote SNPB.

Additional Comments: When the remote SNPB receives this primitive, from the PNB, it can remove the data unit from its queue.

5.8.2 Connectionless SNPB Connection Reference Structure

The connection reference table and its entries are similar to the connection-oriented subnetwork case. However, in the connectionless case, the FIFO queue is specified instead of the Window queue. It may contains the following information:

Addresspair: Source and destination address,

ConInfo: Various information, such as: subnetwork information, size of buffer for data unit queues, etc.

Ques: FIFO queues for data transport.

5.8.3 Connectionless SNPB Elements of Procedure

5.8.3.1 General

This section specifies the individual communication entities of the connectionless SNPB. In the generic gateway, the connectionless subnetwork SNPB also maintains connection reference (ConRef) and all service primitives are referenced by ConRef.

6.7.3.2 Connection Establishment Phase

The first data unit between a pair of communication nodes, sender and receiver, initiates a connection reference (ConRef). The connection may be used for the data unit's transfer between those nodes. The connection reference for the connectionless subnetwork yields advantages as follows: 1) the uniform interface is possible between SNPB and PNB; 2) if the remote subnetwork is connection-oriented subnetwork then the connection entry can be easily located between local and remote SNPB.

The connection establishment may be initiated by local subnetwork user or remote subnetwork user.

a) By local subnetwork user:

When the SNPB receives a DataIndM primitive from the SMAB, the connection reference table is searched whether the connection reference is available or not with the address pair (sender and receiver) of the primitive. If there is an connection reference for the address pair then the data unit can be transferred with the connection reference to the PNB. On the other hand, if there is no connection reference for the address pair in the connection reference table, then the primitive is assumed as the first data unit between the communicating pair nodes. Then, a new connection reference will be created and any further data units may be referenced between those pair nodes. In either case, the data unit will be sent to the PNB as DataIndP primitive.

b) By remote subnetwork user:

Unlike the connection establishment by the local data unit node, the unreferenced data unit is known by the SNPB. When a data unit is transferred from the PNB the data unit is already linked to a connection reference entry or it is referenced to an empty reference. When the reference is 'S0', this implies that the data unit is not a referenced data unit and a new connection reference may be created with the address pair which carried on the 'S0'.

5.8.3.3 Data Transport Phase

In this specification, two FIFO queues are used for the data unit storage, one for each directional data transport. The FIFO queue is one of the well known temporal storage management scheme for the communication systems (see Figure 5.12). It includes two basic operations, "insert" and "remove." The "insert" adds a block of object(data) in the queue right after the last inserted object block. The "remove" extracts the oldest object block which was inserted in the queue.

```
         Head                                        Tail
       +---------------------------------------------+
  -->  |          ---->       --->        ---->       |  -->
       +---------------------------------------------+
(Insert Y)                                        (Remove)
```

Figure 5.12 FIFO Queue

The data transport operation is quite similar to the connectionoriented subnetwork case. However, the queue management is different, such as in the window queue to FIFO queue.

5.8.3.4 Connection Termination

Because the SNPB does not represent the connection-oriented subnetwork, the connection termination cannot explicitly initiated by a local user. Instead, the inactive state for reasonably long periods may cause the connection termination. The long inactive state may have two causes. First, the connection entities do not produce the data units fast enough such as a terminal-to-host communication. Second, the communication link is experiencing troubles such as loosing data units, or the data unit cannot reach the destination.

5.9 PNB Specification

In the traditional half gateway design approaches, the PNB is a protocol translation module between two or more subnetwork protocols. However, in the generic gateway each subnetwork protocols are already converted into the universal structure and functions which can be interfaced to the PNB. The two interconnected subnetworks may not be inter-operable with each other because of the following reasons: 1) the universal service functions are chosen from the super set of the subnetwork protocol functions. Hence, some service primitives do not exist on both subnetworks; 2) each subnetwork may have different characteristics and requirements. As the result, the PNB is introduced in the generic gateway to provide a resolution on the subnetwork's incompatibility. The functions are called protocol negotiations.

5.9.1 PNB Services

As stated in section 5.5, the PNB functions are not peer protocol functions since the services provided by PNB are not protocol services. Basically, there are two types of PNB services: link services and negotiation services.

5.9.1.1 Link Services

The link service provides a logical interconnection between two communicating SNPBs. For instance, an internetwork user A on subnetwork A wants to communicate with user B on subnetwork B, and a gateway resides in between subnetwork A and subnetwork B. Then, the connection requesting user 'A' sends a connection request to the gateway. The subnetwork A side of SNPB sends the connection indication to the PNB. The connection request is transferred to the subnetwork B. Finally, the connection request is transferred to the user B on the subnetwork B (Figure 5.13). In the same way, other service primitives will be transferred between interconnected subnetworks. The PNB manages the logical interface between subnetwork protocol primitives. For instance, when one of the subnetworks is connection oriented subnetwork and the other subnetwork is a connectionless subnetwork, then the connection request must be represented and transferred as a data unit in the connectionless subnetwork and vice versa.

```
 User A          SNPB        PNB         SNPB          User B
                   ConInd  +-------+  ConReq             ^
  | ConReq       +--------->|       |------->+       ConInd|
+----V----+    +--|---+    +-------+     +--|---+   +---|---+
|         |    |      |                  |      |   |       |
|         |    |      |                  |      |   |       |
+--|-----+     +--|---+                  +---|--+   +---|---+
   V ConReq       | ConInd         ConReq V      ConInd |
   +-------------+                        +------------+
```

Figure 5.13 Connection Request Service

5.9.1.2 Negotiation Services

The PNB provides two types of protocol negotiations. First, there are static negotiations. The static negotiation will be performed with each subnetwork's information which is exported from each SNPB during the gateway initialization step. The results cannot be altered during individual protocol services, or communication sessions. If the result of the static negotiation is a success, then the two subnetworks can communicate with each other without degrading internetwork services. If minor fixes are required, then the two subnetworks will be interoperable with minor functional limitations during operation. However, if the result is failure then the gateway will halt and no further operations are possible. The static negotiation case is shown here.

```
SNPBAInit  ---------->|                  Static        +----------->
             NetAInfo |                  Negotiation   | Success
                      |==> PNBInit  --------------+
                      |                            |
SNPBBInit  ---------->|                            +-----|||
              NetBInfo                                  Fail
```

The second class of negotiation, identified as a dynamic negotiation, is a connection negotiation, which may be applied by individual connection entities. The dynamic negotiation is completed by two phases: 1) during the connection initiation; 2) and during the connection completion:

```
           Initial                     Secondary
Connection Negotiation  Connection  Negotiation   Connection
 Request   ------------> Response  -------------> Completed
```

More specifically, information on this dynamic negotiation will be provided in the following sections.

5.9.2 PNB Connection Reference Structure

To identify the connection for each service primitive, the connection reference table entry must be maintained in the PNB. While each SNPB maintains individual ConRef for each connection entries, the PNB links the ConRefs of each SNPBs. For example, a connection between subnetwork A and subnetwork B is referenced as ConRef A and ConRef B on SNPB A and B. Then, the two ConRefs are linked by PNB as PconRef. Figure 5.14 shows the case. Each subnetwork block (SNPB) doesn't need to understand about the other side's of connection status or information.

```
                    +---------------+
                    | PconRef       |
                    +---------------+
                        |      |
           +------------+      +------------+
           |                                |
       +---------+                      +---------+
       | ConRef  |                      | ConRef  |
       | A       |                      | B       |
       |         |                      |         |
       |         |                      |         |
       |         |                      |         |
       |         |                      |         |
       +---------+                      +---------+
  Connection Reference            Connection Reference
      SNPB A                          SNPB B
```

Figure 5.14 PNB Connection Reference

5.9.3 PNB Elements of Procedure

5.9.3.1 General

This section specifies the individual communication entities on the PNB. The PNB functions are basically decomposed into three classes: 1) SNPB primitive linking; 2) connection establishment between subnetworks; 3) protocol negotiation. However, the functions are not easily separable from each other. Each functional element will be specified by the chronological steps on each communication entity, such as connection establishment, data transfer, and connection termination phase.

5.9.3.2 PNB Initialization Phase

During the gateway, initialization or specific reset state of the gateway, the PNB initialization phase is performed. During this phase the PNB receives all subnetworks information from the SNPBs. With this information, the initial subnetwork protocol negotiations are performed. The information which is passed from the SNPB supposedly is not altered during the operation of the gateway, and the negotiation in this stage is called static negotiation.

Static negotiation doesn't involve any protocol service primitives or any communication entities. It only depends on the subnetwork characteristic information. However, the negotiation at this stage may be critical for overall gateway operation. The failure of the negotiation may result in a total failure of the gateway operation. The negotiated protocol elements are decomposed into two groups: 1) Parameters and 2) Functions.

A few examples of the subnetwork protocol parameters which may be used in the static negotiations are:

a) Time

- Min. packet delivery time,
- Max. packet delay time,
- Min. connection time,

- Max. connection allowed time,
- Max. allowed idle connection time.

b) Packet Size

- Max. allowed packet size,
- Number of packets per messages.

c) Quality of Service,

d) Queue Size.

A few examples of the subnetwork protocol functions which may used in the static negotiations are:

a) Connection establishment

- two way hand shaking,
- three way hand shaking.

b) Data transmission

- require acknowledgement,
- no acknowledgement required,
- windowed sequence control
- FIFO sequence control,
- blocking,
- segmenting.

c) Connection Termination Scheme

- one way
- two way (requires confirm).

The result of the static negotiation which is performed by procedure PNBInit may result in one of three situations: 1) interoperable; 2) interoperable with minor fixes; 3) non interoperable. When the result is either (1) or (2), the gateway can continue the process to perform the internetworking functions. However, if the result is (3) the gateway may give up its function as a internetworking unit. For example, if one of the subnetwork is local area network which allows to wait for the acknowledgement for a packet which was sent with maximum delay of milliseconds and the other side of subnetwork is a long haul network and its average turn around time is 30 milliseconds then the two subnetworks have little hope to be interoperable. The result may not be satisfactory unless there some special care be provided.

5.9.3.3 Connection Establishment Phase

The connection establishment phase is responsible for establishing the logical link between two communication entities. This phase will be activated by receiving a service primitive which can not found in the reference table. More specifically, when a service primitive is received from one of SNPBs and the ConRef of the primitive is not found on the PNB connection reference table, then it may be the connection initiating primitive. The connection initiating service primitives are ConIndP, and DataIndP. When the connection initiating subnetwork is a connection oriented subnetwork, then the ConIndP is applied. When the connection initiating subnetwork is a connectionless subnetwork, then DataIndP is applied.

As soon as the PNB receives the connection initiating primitive from one of the SNPB, it performs the initial negotiation. The initial negotiation is performed with the static subnetwork information and the connection initiating service primitive. This scenario is defined as follows:

```
PNBProtocol [a,b,ds] (infa, infb: SNetInfo) exit :=
  choice con:PConRef [] [con IsIn cons] ->
  (
   (* REQUEST FROM SIDE A *)
   (  [nettype(infa) = connection_oriented] ->
      ( ConIndP[a](con |pkt);
        Initial_Negotiation(infa, infb, con |result);
        (    [result = success] => ...
              []
             [result = fail] =>
              DisReq[a](con);
              ds !con;exit
        )
     [nettype(infa) = connection_less] ->
        DataIndP[a](con |pkt);
        Initial_Negotiation(infa, infb, con |result);
        (    [result = success] =>  ...
          []
           [result = fail] =>
              ds !con;exit
        )
   )
 []
   (* REQUEST FROM SIDE B *)
   ...
endproc (* PNBProtocol *)

process Initial_Negotiation(infa, infb:NetInfo, con:ConRef):
                                 exit(Bool,ConRef) :=
      i:(infa,infb,con !result);

      exit(result)
endproc.
```

If the initial negotiation is successful, then the proper service primitive will be transferred to the connection responding SNPB. The response of the connection request may returned as a ConResP for connection oriented subnetwork and AckIndP for a connectionless subnetwork. When the PNB receives those service primitives the second negotiation will be performed.

```
process WAIT_CONFIRM [a,b,ds] (infa, infb: SNetInfo,
                                            con:PConRef):=

   (  ConRes[b] (con |con');
      Second_Negotiation(infa, infb, con'|result, con);
      (
         [result = success] -> ...
        []
         [result = fail] ->  ...
       )
  []
     .....
endproc.

process Second_Negotiation(infa, infb:NetInfo, con:ConRef):
                                   exit(Bool,ConRef) :=
        i:(infa, infb, con |result);
   exit(result)

endproc
```

The initial negotiation is a negotiation between the connection initiating node and the gateway, and the secondary negotiation is a negotiation between the connection responding node and the gateway. Communication on the connection link is possible when both negotiations terminate successfully. If it terminates unsuccessfully, then the connection will be released and no further communication may be possible with that specific connection reference entry.

There are various connection establishment schemes. However, in this specification only two schemes are specified. Those are acknowledged data unit transmission on connectionless subnetwork and two way handshaking scheme on connection oriented subnetwork. The connection establishment may involve the following sequence of service primitives between SNPBs and PNB.

```
(Source is Conn. oriented)   (Destination is conn. oriented)

Init
   ---> (-)CR ---> (+)CR ---> (-)CC ---> (+)CC --->Connected
 /|\          | (1)     |          | (2)      |
  |         *\|/      *\|/       *\|/       *\|/
  +---------+---------+----------+----------+
                     (+) DR

                      (a)
```

```
(Source is Conn. oriented)   (Destination is connectionless)

Init
   ---> (-)CR  ---> (+)Data --> (-)ACK ----> (+)CC -->Connected
 /|\         | (1)      |         | (2)        |
  |        *\|/       *\|/      *\|/         *\|/
  +--------+----------+---------+----------+
                      (+) DR

                       (b)
```

```
(Source is Connectionless)  (Destination is Conn. oriented)

Init
  ---> (-)Data ---> (+)CR --> (-)CC ---> (+)ACK --> Connected
 /|\         | (1)    |         | (2)     |
  |      *  \|/     *\|/      *\|/      *\|/
  +--------+----------+---------+----------+

                       (c)
```

```
(Source is Connectionless)   (Destination is connectionless)

Init
   --> (-)Data --> (+)Data --> (-)ACK ---> (+)ACK -->Connected
 /|\         | (1)     |         | (2)          |
  |        *\|/      *\|/      *\|/           *\|/
  +--------+----------+---------+------------+

                     (d)
```

Figure 5.15 Connection Establishment Phase (continued)

(-) Receive SNPB Service Primitive
(-)CR: ConIndP, (-)CC: ConResP
(-)Data:DataIndP, (-)Ack: AckIndP

(+) Send SNPB Service Primitive
(+)CR: ConReqP, (+)CC: ConCnfP
(+)Data:DataReqP, (+)Ack: AckReqP
(+)DR: DisReq

(1) Initial Negotiation
(2) Secondary Negotiation

*: Any incomplete connection causes such as
- timeout,
- Disconnection request,
- negotiation failure
- etc.

Figure 5.15 Connection Establishment Phase

5.9.3.4 Data Transfer Phase

The data transfer phase of the PNB is straight forward. This is because the flow and error control are expected to be provided by the subnetwork (SNPB). The data unit transfer from one SNPB to the other SNPB can be done by receiving a data unit from one of SNPB as a DataIndP, and the data unit is sent to the other SNPB as a DataReqP. Then the PNB refuses to receive any further data units unless previously sent data unit is acknowledged from the data unit destination side SNPB. When the acknowledgement of the data unit is received as an AckIndP from the data unit destination SNPB, it is sent to data unit source SNPB as an AckReqP. The PNB is ready for next data unit transfer and so on. The data transfer cycle is as:

DataIndP ---> DataReqP ---> AckIndP ---> AckReqP.

5.9.3.5 Connection Termination Phase

When PNB receives a connection termination service primitive from one of SNPBs, the primitive will be transferred to the other SNPB and the connection reference will be removed. However, a connection with connectionless subnetwork will experience some difficulty in termination, because there are no explicit connection termination service primitives on the connectionless subnetworks. As the result, the number of connection references will grow indefinitely. This condition is very unpleasant especially when the two SNPBs are connectionless subnetworks. In one solution of this difficulty, the connection may be removed by a time-out condition. When there are no interactions between communication entities for specific period, the connection may be assumed broken or terminated and the connection reference may be released.

CHAPTER 6

GENERIC GATEWAY TESTING BY CLIPS

6.1 Introduction

During the last decade, many protocol verification techniques have been developed and studied. However, most verification techniques are limited to the simple protocols. Real protocol verification has not been achieved. One of the most significant limitation on the tradition verification techniques is caused by the exhaustive methods which they relied on. For instance, if there is a simple protocol which requires 10 state transitions from initial to final state, and at each state there are two possible next states. The total number of possible paths between initial to final state become 1024. If the protocol involves time dependency, then verification become almost impossible, because the time constraint may not be a bounded property. In this section, an alternative verification technique is proposed which is called a Verifiable Testing (VT) Scheme.

As Groz [GRO 87] stated, the simulation may not prove the total protocol correctness by any means. However, it may provide enough confidence to the protocol designer. In most cases, the confidence is enough during protocol design phase, especially when no total verifications are possible due to the complexity of the protocol. The scheme VT uses a simulation techniques during verification.

During the VT study, some beneficial points of simulation are discovered. They are:

1. The verified system's behavior can be monitored in real-time. A specifier can easily understand what actually happens when the system is incorrectly specified. The result is that solutions of the problems can be easily sought.

2. The model of the system is an executable system so that any modification on the specification can be directly examined.

3. It does not rely on the exhaustive state exploration by which verification of a complex system is possible.

4. The protocol specifier can expect short turn around time during the refinement of the system.

The CLIPS is used as a VT Tool. CLIPS was developed by the Mission Planning Group at NASA, Huston. CLIPS stands for 'C' Language Production System. A short introduction of the CLIPS is introduced in the following section.

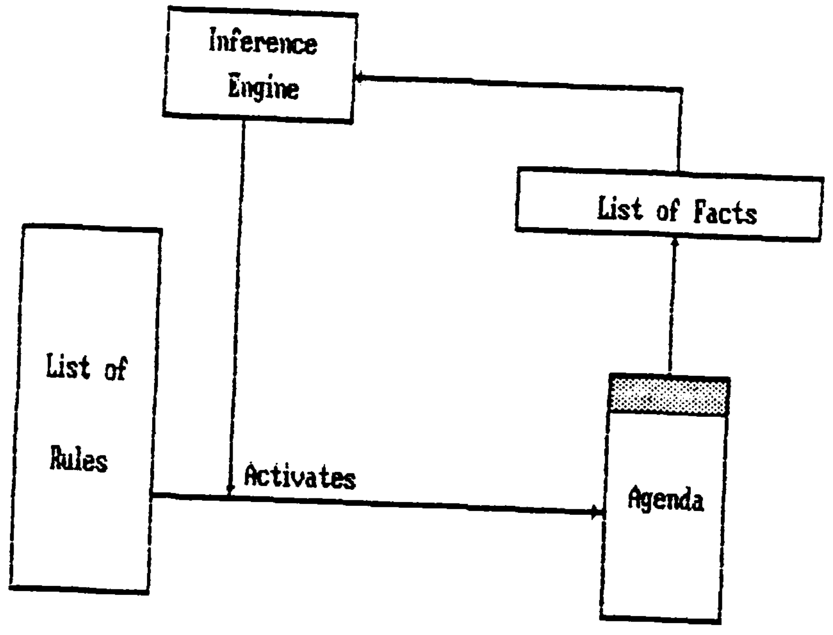
In the VT method, the verification is based on the mechanism for observing internal states and interactions in the system simulation. The verification is performed by one or more processes called testers or observers, that run concurrently in the system.

From our work, CLIPS was found adequate to test the most important communication system's properties. These properties included the following:

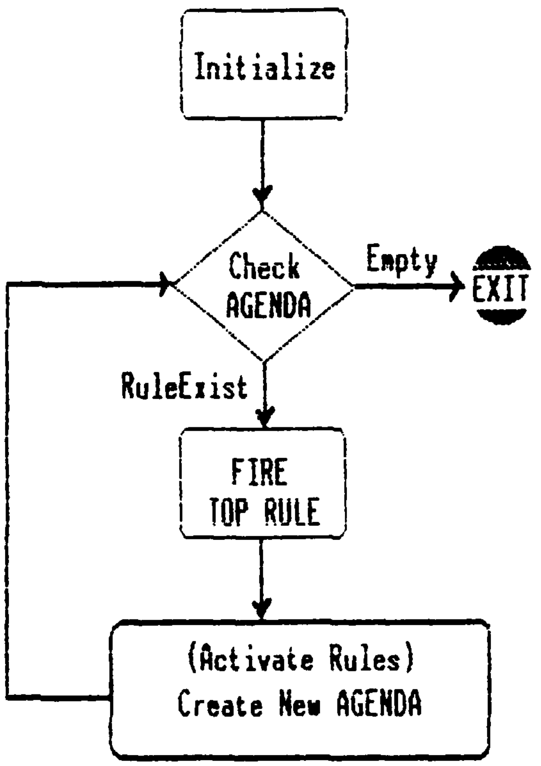
a) Non procedural characteristics of CLIPS are adequate to represent the communication system's characteristics such as concurrent, non-deterministic, and synchronous communication.
b) Testing the invariant property of a system is fairly easy.
c) Deadlock or unspecified specification can be detected.
d) Conversion of the LOTOS specification to CLIPS is almost straightforward.
e) CLIPS is an executable language.
f) Some intelligence can be easily implemented and tested which is desirable for the generic gateway.

6.2 Introduction to CLIPS

The CLIPS has two major properties, rules and facts [CLIPS 87a] [CLIPS 87b]. Each fact represents a piece of information. The rule is an atomic entity which has a condition part, left hand side (LHS), and an action part,



(a)



(b)

Figure 6.1 CLIPS Structure

right hand side (RHS). When the condition part satisfies the rule become activated and fired. The basic execution cycle is as follows [CLIPS 87a] (see Figure 6.1):

a) The facts list is examined to see if any rules conditions have been met.

b) All rules whose conditions(LHS) are currently met are activated and placed on the agenda (list of active rules). The agenda is essentially priority stack.

c) The top (highest priority) rule on the agenda is selected and its RHS actions execute. As a result of RHS actions, new rules can be activated of deactivated. This cycle is repeated until all rules that can fire have done.

The CLIPS is modified (from CLIPS V.4.1) to create a suitable environment for the discrete event simulation. One of the most important modification is in a time dependency on facts. With the modification of CLIPS, the facts can be created with or without delay. If facts are created without delay, the facts can be used immediately. If facts are created with delay, the facts only can be used after the specified delay. Currently usable or accessible facts are called as active facts and not usable facts are called as delayed facts. The delayed facts are created by 'delay-assert' operation with a delay value. The delay value means that the system requires the specified time delay to produce the fact. The delayed facts will be active facts after specified time delay.

The modified basic execution cycle is as follows (see Figure 6.2):

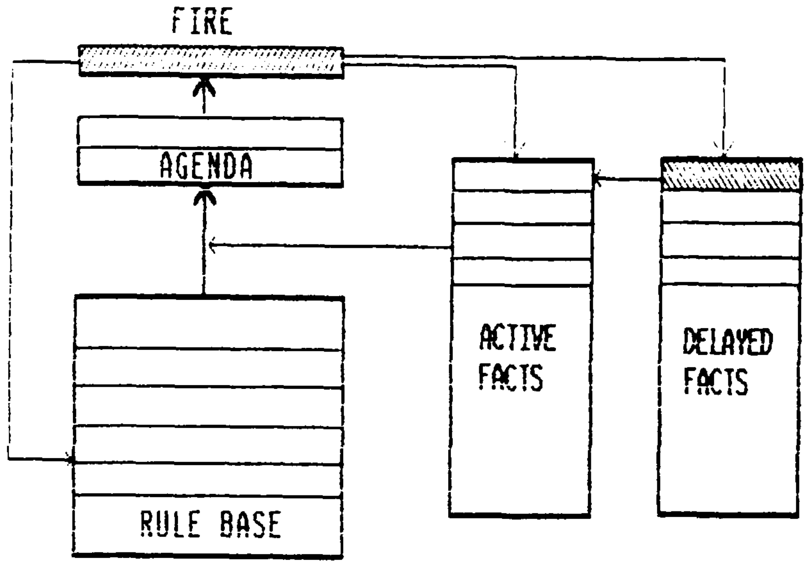
a) The facts list is examined to see if any rules conditions have been met.

b) All rules whose conditions(LHS) are currently met are activated and placed on the agenda (list of active rules). The agenda is essentially priority stack.

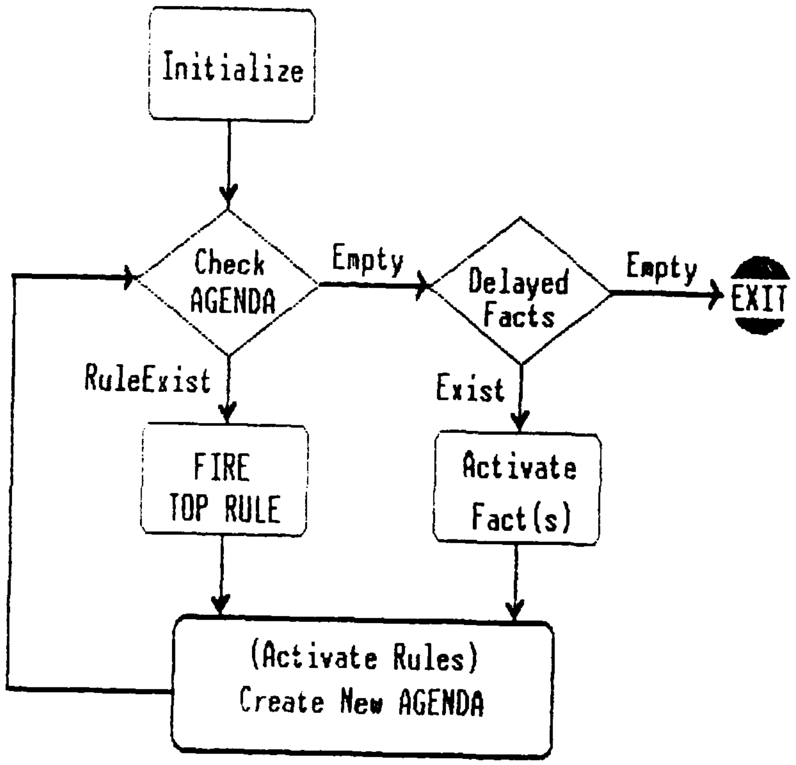
c) The top (highest priority) rule on the agenda is selected and its RHS actions execute. As a result of RHS actions, new rules can be activated of deactivated.

d) If the AGENDA is empty, then one or more delayed facts are activated (become active facts) until any rules can be activated or the delayed facts list become empty.

e) This cycle is repeated until all rules that can fire have done and no more delayed facts available.



(a)



(b)

Figure 6.2 Modified CLIPS Structure

6.2.1 New functions

Various modifications and extensions were made to CLIPS. First the following capabilities are eliminated; embedded editor, extended math functions, and online help utility. Secondly, the code has been recompiled to make a more compact code. As the result more free memory space become available for the testing environment which is critical in the PC environment. A a few new functions are implemented. They can be grouped as, window manager routines, extended list processing functions, and delayed assertion. These are described below.

6.2.1.1 Window management

The window management has been implemented on the original NASA version of CLIPS (V. 4.1). However, entire window management routines are implemented independently, for the following two reasons. First, when this project was started the source code of the window routines was not available. The compiled version of the CLIPS did not support the proper user interface. Secondly, our window routines takes less memory and easy to use. The window management functions are presented as bellow:

Note: Variables can be identified by a symbol '?' or '$?'. When the variable name starts with '?' it means that it is a single valued variable. When the variable name starts with '$?' it means that it is a multiple valued variable (List).

(clear-screen) - Erases entire screen.

(clear-window) - Erase specified window area.

(window-on), (window-off) - Activates or deactivates window dependent routines.

(set-window <?name> <?color> <$?window-area>) - Creates a box around window area with a window name on top left corner of the window.

<?name> - Window name and it can be used as a window object identification.

<?color> - Background color of the specific window. Legal colors on PC environment are:
BLUE, RED, GREEN, BLACK, WHITE, YELLOW.

<$?window-area> - Boundary of the specific window and it must be multi-valued variable. It must have exactly four number valued elements. The each element represents screen coordinates (x-min, y-min,x-max, y-max) of window region.

ex: (set-window Sender BLUE $?my_win)

(wprintout <?field> <?linepos> <?string> <?name>) - Prints at most two elements at the <?linepos> on the window, first one is a field name and the second one is a content of the field.

<?field> - It can be used as a field name on the window. This field may be suppressed by using empty string ("").

<?linepos> - It specifies a line position for the text in that specific window. It must be integer type.

<?name> - Identifier of the window object. The name must be pre-defined by 'set-window' command.

ex: (wprintout Send 1 "Data Request" Sender) prints "Send: Data Request" at 1st line of the specified window "Sender".

(move-to <?x-pos> <?y-pos>) - Moves current cursor to new location.

<?x-pos> - x coordinate on the screen.

<?y-pos> - y coordinate on the screen.

ex: (move-to 0 0) moves cursor at top left corner.

(show-windows) - It prints all window objects which are defined by 'set-window' command. An example of output format is:

```
(Sender 1 1 10 10)
(Receiver 40 1 70 10)
2 Windows are defined
```

6.2.1.2 List management routines

CLIPS has various useful list management functions. However, some difficulties have been experienced. The following new routines are implemented as a result.

(get-value <?Key> <$?list>) - When the <?Key> is found in <$?list> it returns an element which follows <?key> in <$?list>.

<?Key> - Key for searching value.

<$?list> - multiple valued variable.

Ex: Suppose current multiple variable $?packet has elements, ($?addresses $?options DATA msg $?other).

(get_value DATA $?packet) returns 'msg'

Note: The "get_value" is useful when a specific information needs to be extracted without specifying the exact location of the information in the list. In this specific example content of data field (msg) in the packet is extracted by a key word "DATA".

(add-atom <?elm> <$?list>) - It appends new element <?elm> in the <$?list>. It appends <?elm> only if <?elm> is not pre-exist in the <$?list>.

<?elm> - New element to be inserted.

<$?list> - multiple valued variable (list).

Ex: Suppose a list '$?colors' has (red blue black).
(add-atom green $?colors) will changes the
contents of $?colors into (red blue black green).

(rem-atom <?elm> <$?list>) - It removes <?elm> from
<$?list> only if it exist in <$?list>.
<?elm> - Element to be removed.
<$?list> - multiple valued variable (list).
Ex: Suppose a list '$?colors' has (red blue black).
(rem-atom blue $?colors) changes the contents
of $?colors into (red black).

(swap-atom <?first> <?second> <$?list>) - Exchanges the value
of elements between <?first> and <?second> position in
<$?list>.
<?first>, <?second> - number valued elements but 'end'
can be used to identify the last position of the
<$?list>.
Ex: Suppose a list '$?colors' has (red blue black).
(swap-atom 1 end $?colors) changes the contents
of $?colors into (blue black red).

(mstr-cat <element> {<element>..}) - It concatenates two or
more elements together and returns as a single string.
The each elements are separated by single space.
<element> - It may single or multiple valued variable.

Ex: Suppose a list '$?colors' has (red blue black).
(mstr_cat Color List: $?colors) returns with a
string type value 'Color List: red blue green'.

6.2.1.3 Storage management

In CLIPS, the data storage elements are limited to facts. In many occasions, the facts are not suitable for the VT operation. For instance, the following buffer history invariant test rule may be locked in an infinite loop.

```
(defrule Buffer_History_Test
   (Packet ?data)
   ?x <- (Buffer $?list)
=>
   (retract ?x)
   (assert (Buffer $?list ?data))
```

From the above example, the 'Buffer_History_Test' creates a new fact (Buffer $?list) during its execution. The fact activates 'Buffer_History_Test'. And the rule creates a new (Buffer $?list) and so on. As the result the 'Buffer_History_Test' will be locked in an infinite loop. As a solution of this problem a new storage management method is provided. Two types of storage elements are provided 'record' and 'var'. The storage type identified as a 'record' is a multiple element variable which can be accessed globally. The storage type identified as a 'var' is a single element variable and it also can be accessed globally. With these, the above example may avoid infinite loops. The above example may rewritten as the follow.

```
(defrule Buffer_History_Test
   (Packet ?data)
  =>
   (record Buffer ?data))
```

The new group of functions are as follows.

Multi-valued variable (record):

(create-record <?name> {<element>}) - It creates a list and the contents of the list can be accessible by the name of list.

<?name> - Identifier of the list structure.

{<element>} - Initial elements of the list. Each element may single or multiple valued variable.

ex: (create-record Queue data0 data1)

(record <?name> {<element>}) - It appends <element> to the list.

<?name> - Identifier of the list structure.

{<element>} - Initial elements of the list. Each element may single or multiple valued variable.

ex: (record Queue data2)

(show-records) - prints all records created during program execution and their contents.

ex: (show-records) prints

```
(Queue data0 data1 data2)
1 records are defined.
```

(recall-record <?name>) - It returns the contents of record <?name> as a single string value. The each elements in the list are separated by a space.

ex: (recall-record Queue) returns "data0 data1 data2".

(compare-records <?name1> <?name2>) - It returns the last position where the two records has the same valued elements.

ex: Suppose a record Sender has (data1 data2 Data3) and a record Receiver has (data1 data2 data4) then: return value of (compare-records Sender Receiver) is 2.

(reset-record <?name>) - It frees all elements in record <?name>.

(length-record <?name>) - It returns length of record <?name>.

Single-valued variable (var):

(set-var <?varname> <?value>) - If a variable identified <?varname> is already exist in variable list then put the <?value> to the variable. Otherwise create a storage for the variable and set the variable value to

<?value>.

ex: (set-var Last_Test OK)

(get-var <?varname>) - It returns the value of the variable which identified <?varname>.

```
ex: (if (eq (get-var Last_Test) ERROR)) then
        (printout ERROR))
```

(show-vars) - It prints all currently defined variable names and their values.

```
ex: (show-vars) will prints
     (Last_test OK)
     1 variables are defined
```

6.2.1.4 Time Constraint Related Routines

The following routines are provided to support time dependency to the CLIPS. More detailed description will be given in the section 6.2.2.

(delay-assert <?Object-name> <?delay> <fact>): It creates a fact <fact>. But the fact is not usable unless 'sys_time' is progressed by <?delay>.

ex: (delay-assert Sender 5 DataInd data0) will creates a fact (Packet data0) after 5 units delay.

(show-systime): It returns current value of sys_time.

(show-dfacts): It prints all facts which are created by 'delay-assert' but they are not activated yet.

An example of output format is:

```
(Sender   10 10 DataInd gen10)
(Receiver 11 10 WaitAck)
2 facts are waiting

(Sender   10 10)
(Receiver 10 10)
2 objects are defined
```

The first group shows all facts which are waiting for activation. The first element represents a name of an object which creates the fact. Second and third elements represent the activating time of the fact and base time of the object. The remainder are fact values. The second group shows all objects which are created. First element represents a name of object. The other two elements represents a global time and local time of each object.

6.2.1.5 Other Functions

(restart) - The function is a combined functions of (release -mem), (reset), and (run).

(watch rules) - It prints currently fired rule name. When window flag is on the rule name is printed on left bottom of the screen.

(watch-agenda {on/off}) - It is only effective when the window flag is on. A window named 'AGENDA' is created on right bottom corner on the screen. The window contains all rule names in the agenda (activated rules list).

(rand) - It returns with a random number.

Ex: (bind ?new (rand))

(max <element> {<element>}) - It returns the largest value.

Ex: (max 1 4 $?nums) will returns 4 if all elements of $?nums has smaller value than 4. Otherwise it will returns the value of largest valued element in $?nums.

(min <element> {<element>}) - It returns the smallest value.

Ex: (mix 1 4 $?nums) will returns 1 if all elements of $?nums has larger value than 1. Otherwise it will returns the value of smallest valued element in $?nums.

Another interesting modification on the CLIPS is a speed control capability. If window flag is on by the "(window-on)" command the following key strokes will change the speed of the execution.

S: slows down the speed of execution.

F: increases the speed of execution which reduced by the 's' keys.

C: returns to full speed.

<space>: changes the execution to single step mode. And any key input except 'c' leads one cycle of rule execution.
The single step operation will be returned to full speed by 'c' key stroke.

6.2.2 Delay Assertion

When a function 'delay_assert' is called, a new fact is created but it is not linked to the pool of facts (facts list). Instead, it is stored in the temporary list until no rules are found in agenda. If there no rules in the agenda, then one or more delayed facts will be linked to the main facts list until any rule can enters to the agenda or the temporary list become empty. The temporary list is maintained by the chronological order to make sure that the facts are activated in order. Since the communication systems involves more than objects which operate in parallel, each object may requires to maintain its own time base. The time base must be updated independently. The time bases of objects must be synchronized to a global clock. To aid in the understanding of the delayed assertion function, a simple communication system is provided bellow.

Suppose there are two objects called 'Sender_A' and 'Sender_B' and one object called 'Receiver'. The senders create and send a series of packets to the 'Receiver'. The 'Receiver' receives the packets from both senders. Now 'Sender_A' is responsible to send a series of packets at every time interval but 'Sender_B' is responsible to send a series of packets at every other time interval. Then the simple communication system can be represented as bellow.

```
(deffacts Test
   (senderA)
   (senderB)
   (receiver))

(defrule senderA
   ?f <- (senderA)

  =>
   (retract ?f)
   (assert (data A))
   (delay-assert SenderA 1 senderA))

(defrule senderB
   ?f <- (senderB)

  =>
   (retract ?f)
   (assert (data B))
   (delay-assert SenderB 2 senderB))


(defrule receiver
    ?f <- (data ?src)
    (receiver)
 =>
    (retract ?f)
    (printout "DATA Received from " ?src crlf))
```

The interactions between two senders and a receiver are listed in Figure 6.3. The first field represents timer values. First column labeled as 'S' is a global system time. And second and third columns which labeled as 'A' and 'B' are local time on sender 'A' and 'B' respectively. The second field represents a delayed facts list which temporary stores any facts created by 'delay-assert' function. The value '(A4)' means a fact "(SenderA)" which suppose to be activated when the system clock reaches 4. The third field represents content of agenda, viz. list of currently activated rules. The fourth field represents a name of currently fired rule. 'A' means SenderA and 'B' means SenderB. The last field represents the action of the receiver when current rule is fired, in this case the received packet is represented.

The initial facts which are created by deffacts 'Test' (which defines facts), two rules 'senderA' and 'senderB', are activated and fired. Mean while, senderA and senderB generate facts (data A) and (data B). The facts activates a receiver and the facts are consumed by the receiver. Because the facts are immediately available there no progress on system clock has been made. Then, the AGENDA becomes empty. During the rules execution two delayed facts are created which has delay value 1 and 2 by senderA and senderB. The empty AGENDA forces to activate a delayed fact (data A) which has smallest time value. Then the system clock is updated to 1. During the system clock is 1 the rule 'senderA' is inserted in the AGENDA and fired by the activated fact (senderA). The rule generates a message fact (data A) and delayed fact (senderA). Like the previous step the fact is consumed by a rule receiver and the (senderA) is inserted in the delayed facts list. Again the AGENDA becomes empty. This causes an activation on delayed facts, in this case (senderB) is selected because it has the smallest time value and also it was created before (senderA) which has the same time value. The fact activates 'senderB' and the system time is updated to '2', and so on. As the result, receiver receives a series of packets from 'senderA' and 'senderB'. And the packet generation ratio between 'senderA' and 'senderB' can be maintained by 2:1.

More detailed CLIPS will not be discussed in this paper however they can be found from [CLIPS 87a] and [CLIPS 87b].

S	A	B	(Qa)	(Qb)	Agenda	Fired	Received
0	-	-	-	-	(a,b)	-	-
0	-	0	-	(B2)	(a)	B	(Data B)
0	0	0	(A1)	(B2)	-	A	(Data A)
1	1	0	-	(B2)	(a)	-	-
1	1	0	(A2)	(B2)	-	A	(Data A)
2	1	2	(A2)	-	(b)	-	-
2	1	2	(A2)	(B4)	-	B	(Data B)
2	2	2	-	(B4)	(a)	-	-
2	2	2	(A3)	(B4)	-	A	(Data A)
3	3	2	-	(B4)	(a)	-	-
3	3	2	(A4)	(B4)	-	A	(Data A)
4	3	4	(A4)	-	(b)	-	-

Figure 6.3 Delay Assertion

6.3 Verifiable Test Scheme

The verifiable test system has a three level hierarchical structure. From the base, there is a model of tested system, a tester, and a verifier. The model represents the system's behavior. The tester monitors the model, it also referred to as an observer. One of the important behaviors of the tester is in the disjointed operation to the tested system. This means that the tester should not disturb the behavior of the tested system by any means; such as any shared variables or messages should not be allowed between model and tester. Finally the verifier can be either a set of processes or a human verifier. It is responsible for the followings:

1. Validation of the result of test process.
2. Monitoring the misbehavior of the system when it is encountered.
3. Management of the test environment.
4. Analysis of the misbehavior of the system.
5. Diagnosis on the causes of misbehavior of the tested model.

The importance of the verifier forces us to select an AI language as a VT tool. The relation between the three level is illustrated Figure 6.4. As an example of the VT, Alternating Bit Protocol (ABP) verification is given in the following sections.

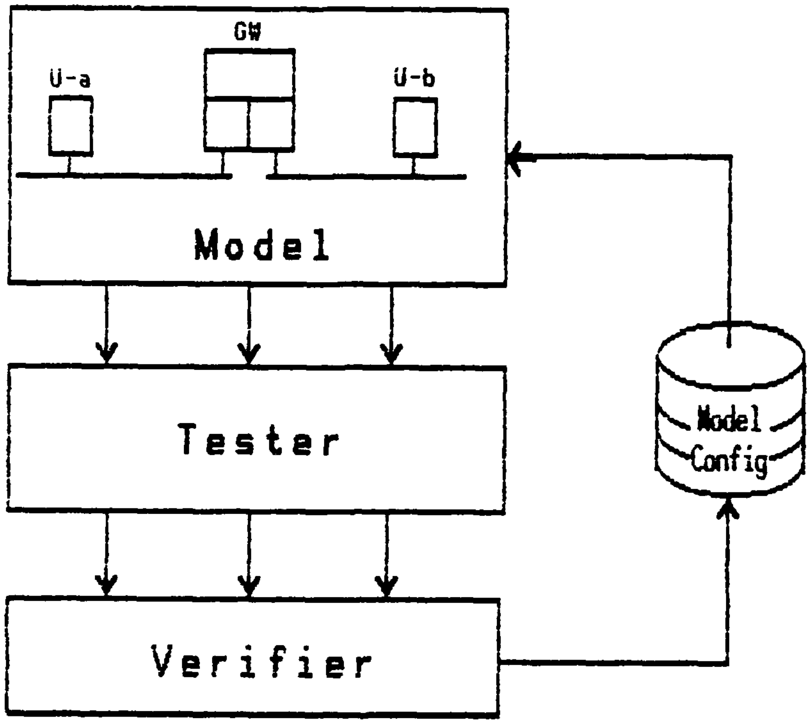


Figure 6.4 Hierarchical Structure of VT System

6.4 Alternating Bit Protocol

Alternating Bit Protocol(ABP) is a simple communication protocol between a pair of nodes, a sender and a receiver. The communication by ABP is unidirectional, such as only sender node is allowed to send a series of messages. The flow and error recovery is controlled by a single bit information on each nodes. Which called a 'bit' and an 'ack' at sender and receiver respectively. The bit value '0' implies that the sender is ready to send or in process of an even ordered message block. The ack value '0' implies that the receiver is expecting to receive an even ordered message block. As the same way the "bit", "ack" valued '1' implies that the sender and receiver is currently managing an odd ordered message block.

The formal specification of ABP by Sunshine[SUN 83B] is as below (Figure 6.5). The symbols which ending with '0' implies that it is related to an even ordered message. And the symbols which ending with '0' implies that it is related to an odd ordered message. For example, Sender state S0 generates an even ordered message. Receiver state A0 replies an acknowledgement by notifying that it is expecting to receive an even ordered message and so on.

```
C0::= send0 W0                    S0::= ReceiveMsg0
W0::= ReceiveAck1  S1                    <give  message to user>
A1
ReceiveAck0  C0                        ReceiveMsg1     A0
Timeout  C0                                ReceiveMsgError A0
                                  A0::= SendAck0  S0

S1::= <get message> C1
C1::= send0 W1                     S1::= ReceiveMsg1
W1::= ReceiveAck1   S0                    <give message to user>
A0
          ReceiveAck0  C1                        ReceiveMsg0     A1
          Timeout  C1                            ReceiveMsgError A1
                                 A1::= SendAck1  S1

        (a) Sender                          (b) Receiver
```

Figure 6.5 Alternating Bit Protocol in Formal Grammar

6.4.1 Protocol Model Specification

Conversion from formal specification to CLIPS representation is quite straightforward. In Figure 6.6, some modifications are made to reduce the number of rules. As the result the state S0 and S1 can be represented by a single rule 'sender_gen_pkt' with different 'bit' value, and so on.

The state variable of formal specification is represented as a fact in CLIPS specification. For example the state of sender 'S0' in formal specification can be represented by a fact (Sender GenPkt 0) in CLIPS. If the fact exist the rule 'sender_gen_pkt' will be linked in the AGENDA and it can be fired. During its execution the fact which activates the rule, (Sender GenPkt 0) is removed and new facts are created. They are (SenderBuf ?msg) and (Sender SndPkt 0). The (Sender SndPkt 0) implies that the next state of sender will be 'sender_send_pkt' or 'C0' in formal specification. And (SenderBuf =(gensym)) is a message which will be sent during the next state and its message content is an internally generated symbol. The message passing between sender and receiver is the same mechanism as above. Such as a message created by a sender is a fact which would be retracted by a receiver. For instance a message (DATA ?msg ?bit) is created by the sender in state 'sender_send_pkt' and it is consumed by the receiver in either 'receiver_wait_pkt1' or 'receiver_wait_ pkt2' state.

As noticed, three 'delay-assert' are found in Figure 6.6, from 'sender_gen_pkt', from 'sender_send_pkt', and from 'receiver_ wait_pkt1'. The 'delay-assert' creates a special fact which will be available only after specified time units. For instance (delay-assert Sender ?gentime Sender SndPkt ?bit) implies that the sender takes ?gentime units of interval to produce a message. After the delay the sender will be in the state (Sender SndPkt ?bit). The time dependency of the ABP specification can be achieved by 'delay-assert'.

```
(defrule sender_gen_pkt  " Generate a packet"
    ?fx <- (Sender GenPkt ?bit)
=>
    (retract ?fx)
    (assert (SenderBuf =(gensym)))
    (delay-assert Sender ?gentime Sender SndPkt ?bit))

(defrule sender_send_pkt  " Send a packet "
    ?fx <- (Sender SndPkt ?bit)
    (SenderBuf ?msg)
=>
    (retract ?fx)
    (assert (DATA  ?msg ?bit))
    (assert (Sender WaitAck ?bit))
    (delay-assert Sender ?timeout SenderTimer))

(defrule sender_wait_ack1 "Receive expected ack."
    ?fx <- (Sender WaitAck ?bit ?time)
    ?fy <- (ACK ?ack&:(neq ?ack ?bit))
    ?fz <- (SenderBuf ?msg)
 =>
    (retract ?fx ?fy ?fz)
    (assert (Sender GenPkt =(mod (+ ?bit 1) 2))))

(defrule sender_wait_ack2  "Receive unexpected ack."
    ?fx <- (Sender WaitAck ?bit ?time)
    ?fy <- (ACK ?ack&:(eq ?ack ?bit))
=>
    (retract ?fx ?fy)
    (assert (Sender SndPkt ?bit)))


(defrule sender_timeout "Timeout reached"
    ?fx <- (Sender WaitAck ?bit ?time)
    ?fy <- (SenderTimer ?msg)
=>
    (retract ?fx ?fy)
    (assert (Sender SndPkt ?bit)))
```

(a) Rules for Sender

Figure 6.6 ABP Protocol in CLIPS Representation (Continued)

```
 (defrule receiver_wait_pkt1 "Receive expected msg. block"
   ?fx <- (Receiver WaitPkt ?ack)
   ?fy <- (DATA ?msg ?bit&:(eq ?ack ?bit))
=>
   (retract ?fx ?fy)
   (assert (ReceiverBuf ?msg))
   (delay-assert Receiver ?consumetime
                           Receiver SndAck ?bit)))

(defrule receiver_wait_pkt2 "Receive unexpected msg. block"
   ?fx <- (Receiver WaitPkt ?ack)
   ?tx <- (DATA ?msg ?bit&:(neq ?ack ?bit))
=>
   (retract ?fx ?tx)
   (assert (Receiver SndAck ?ack)))

(defrule receiver_snd_ack "Sends an ack."
   ?fx <- (Receiver SndAck ?ack)
=>
   (retract ?fx)
   (assert (ACK ?ack))
   (assert (Receiver WaitPkt ?ack)))
```

(b) Rules for Receiver

Figure 6.6 ABP Protocol in CLIPS Representation

To make the protocol model realistic, the third communication entity, a channel is introduced. The channel is a communication path between a sender and a receiver. The channel may involves errors such as corrupted or lost message. Also it has a propagation delay. The simplified channel is represented in Figure 6.7. By the introduction of channel entity the structure of the messages need to be changed. Such as (DATA ?msg ?bit) have to be changed to (Snd_DATA ?msg ?bit) at sender, and to (Rcv_DATA ?msg ?bit) at receiver. And also (ACK ?ack) have to be changed to (Snd_ACK ?ack) and (Rcv_ACK ?ack) on sender and receiver respectively.

```
(defrule channel1 "Message channel"
   ?fx <- (Snd_DATA ?msg ?bit)
=>
  (retract ?fx )
  (if (not ?lost)) then
      (delay-assert Channel_D ?delay Rcv_DATA ?msg ?bit))

 (defrule channel2 "Ack channel"
   ?fx <- (Snd_ACK ?ack)
=>
  (retract ?fx)
  (delay-assert Channel_A ?delay Rcv_ACK ?ack))
```

Figure 6.7 Communication Channel

6.4.2 Tester

From previous sections a time-dependent ABP specification with CLIPS is presented. In this section the verification technique with CLIPS will be introduced. Basically the verification involved a list of properties such as functional correctness, safeness and liveness. Functional correctness means that the system provides the services what it is intended to do. Safeness means that no bad thing happen. It includes deadlock and tempo-blocking freeness. Deadlock is a state in which all processes are blocked forever. Tempo blocking or livelock is caused by a nonterminiating loop of states in which no final state can be reached. Another useful property is a Liveness, which states that either some event in the system is enabled or the system is in final state.

6.4.2.1 Functional Correctness

The functional correctness of the ABP protocol can be stated by verifying the following properties:

1. All messages which sent by a sender are received by a receiver.
2. The messages which received by receiver must be in the same order as the sender generated.

For the functional correctness validation the invariant test on buffer history is used. During simulation the tester monitors the message generation and consumption at sender and receiver respectively. The test rules for buffer history are:

```
(defrule Buf_Sender
   (SenderBuf ?msg)
=>
   (record S_Buffer ?msg))


(defrule Buf_Receiver
   ?fx <- (ReceiverBuf ?msg)
 =>
   (retract ?fx)
   (record R_Buffer ?msg)
   (if (> (length-record R_Buffer)
       (compare-records S_Buffer R_Buffer))
   then
       (assert (ERROR History)))
```

The rule 'Buf_Sender' records all messages which generated by the sender on a list of buffers called 'S_Buffer'. The rule 'Buf_Receiver' records all messages which accepted by the receiver on a list of buffers called 'R_buffer'. The rule is also responsible to check whether or not the history buffers 'S_Buffer' and 'R_Buffer' have the same contents.

6.4.2.2 Safeness

Deadlock conditions can be easily detected. If the system reaches a deadlock state then the protocol will terminate at the non final state. Because the ABP is non terminating protocol, an artificial final state must be provided. In our case the terminal conditions will be selected by verifier which will be discussed in the section 6.4.3.

The ABP is a cyclic protocol by that tempo blocking freeness can be validated by showing that in each cycle some productive work be produced. The productive work means a message block transport from the sender to the receiver. In the following rule the packet generating state of sender, (Sender GenPkt ?bit), is selected as a cycle monitoring position. The progressiveness of the ABP can be shown by monitoring the number of messages which transferred from sender to receiver within a cycle. The number of messages must be exactly one. Otherwise there is a tempo blocking error or progressive error.

```
    (defrule Tempo_Blocking
  (Sender GenPkt ?bit)
=>
  (if (neq (length-record R_Buffer)
           (+ (get-var Last_Cycle) 1))
  then
      (assert (ERROR Progressive))
 (set-var Last_Cycle (length-record R_Buffer)))
```

6.4.3 Verifier

While the above tester validates a protocol on a single case of operation, the verifier tries all possible cases. In this specific APB protocol verification, a few timing constraints are used as the validating domain.

1. Sender packet generation delay.
2. Sender timeout value for an acknowledgement.
3. Delay for receiver to consume the message.
4. Communication channel propagation delay.

Each time constraint domain is give by a upper and a lower limit. Within the limit every possible combinations of the time constraints are tested whether or not the system operates properly. The verification by simulation may not be economical solution if the verification requires to validate all cases in the domain. Because it also experiences the explosive number of cases. By that the heuristic or intuition should be used to limit the number of tested cases. For instance in this ABP verification as an example, if a protocol is invalidated with a communication channel delay n, then the protocol is more likely to operate incorrectly with the delay m (n < m). By the intuition above the test case with channel delay m can be eliminated. If verifier has more intuitions or knowledge on the verified system then less efforts are required to verify the system.

Because the ABP is a non-terminating cyclic protocol, one or more terminating conditions must be provided. One simple terminating condition is a number of packets which transferred from sender to receiver. It may be defined as a domain which is bounded by a lower and an upper limited. Secondly, the maximum number of lost message or acknowledgement and retransmissions within a cycle can be used as a terminating condition.

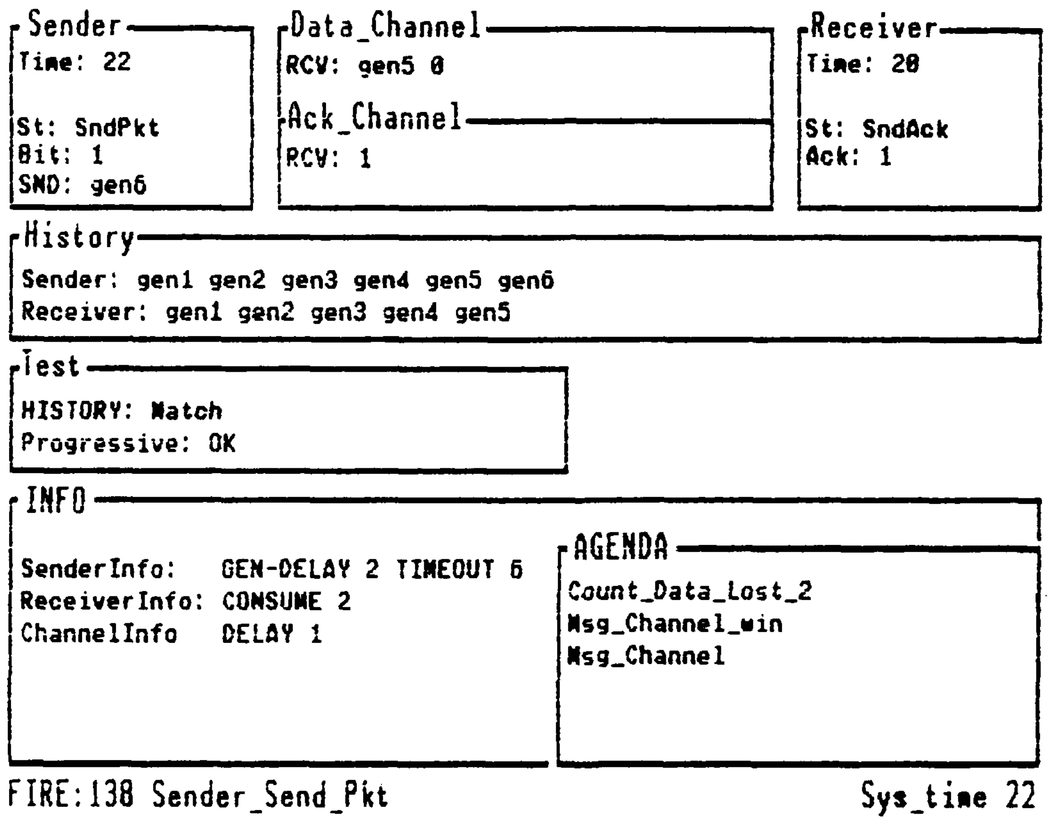
Due to the limited space, the actual inference rules for the ABP will not be provided in this report.

6.4.4 Test result

Two types of results can be expected. First, the result can be observed during test from screen. Secondly, the result can be collected from the log file. One typical screen is illustrated in Figure 6.8. The screen will be partitioned into a number of areas which called windows. First, the current status and actions on each communicating entities can be found from a number of windows which labeled by communication entity names such as 'Sender' and 'Receiver'. In this particular example, the sender window provides the following information: it is in state 'Send_packet'; its current bit value is 1; local time is 22; and the last action of sender was sending a message, gen6. Under the communication entity windows there is a window 'History', which reveals the current history of message of sender and receiver. Current test results are displayed in the 'Test' window. In the window 'INFO' current setup or conditions of test are displayed. On the right lower corner there is a special window called 'AGENDA'. In here all activated rules are displayed by that the next status of the system can be easily predicted. The window 'AGENDA' can be disabled. Finally at the bottom line a name of current rule which fired last time and system clock value are displayed.

The observation from screen is quite useful to understand the behavior of the system. However the results of the tested sessions need to be recorded in a file for the post analysis.

The ABP is proven as robust on timeout or lost in messages and acknowledgements. But, it failed when the channel propagation delay for the message or the acknowledgement are not fixed. The messages are propagated by more than one path which may have different speed. An example of the misbehavior is illustrated in Figure 6.9 which is extracted from test logfile. In Figure 6.9, the sender message generation delay is 2 (units), sender timeout for an acknowledgement is 6, receiver message consume delay is 2, and channel propagation delay is selected randomly between 0 to 5. Due to the timeout for an acknowledgement the sender sends the message (gen1 0) twice at system time 2 and 8. And the receiver receives the first message (gen1 0) and sends back an acknowledgement back (Ack 1). Then the sender sends the second message (gen2 1). The receiver sends an acknowledgement (Ack 0) for (gen2 1). At the moment the resent message (gen1 0) is finally received by receiver because it took more delay than (gen2 1). Erroneously the receiver accepts (gen1 0) for the next expected message, because the receiver waiting for an even ordered message. The protocol failure is detected by buffer history invariant test.



```
Sender              Data_Channel                    Receiver
Time: 22            RCV: gen5 0                     Time: 20

St: SndPkt          Ack_Channel                     St: SndAck
Bit: 1              RCV: 1                          Ack: 1
SND: gen6

History
Sender: gen1 gen2 gen3 gen4 gen5 gen6
Receiver: gen1 gen2 gen3 gen4 gen5

Test
HISTORY: Match
Progressive: OK

INFO
                                            AGENDA
SenderInfo:   GEN-DELAY 2 TIMEOUT 6         Count_Data_Lost_2
ReceiverInfo: CONSUME 2                     Msg_Channel_win
ChannelInfo   DELAY 1                       Msg_Channel

FIRE:138 Sender_Send_Pkt                                Sys_time 22
```

Figure 6.8 ABP Test Screen

```
          ========== ABP TEST =========

SenderInfo:    GEN-DELAY 2 TIMEOUT 6
ReceiverInfo: CONSUME 2
ChannelInfo    DELAY 0 to 5

RESULT: FAIL    Reason: History

  History(Sender)   = gen1 gen2
  History(Receiver) = gen1 gen2 gen1


 2 Sender(0) -----> (gen1 0) ----->
 2                 -----> (gen1 0) ----->
 3                         -----> +(gen1 0) -----> Receiver(0)
 5                      <----- (Ack 1)   <------ Receiver(1)
 5                 <----- (Ack 1) <-----
 8 Sender(0) <----- (Timeout 2)
 8 Sender(0) -----> (gen1 0) ----->
 8                 -----> (gen1 0) ----->
 8 Sender(0) <----- +(Ack 1) <-----
10 Sender(1) -----> (gen2 1) ----->
10                 -----> (gen2 1) ----->
11                         -----> +(gen2 1) -----> Receiver(1)
13                      <-----  (Ack 0)   <----- Receiver(0)
13                         -----> +(gen1 0) -----> Receiver(0)
13                    Lost_Ack (0)  <----

    ------------------------------------------------
```

Figure 6.9 Example of ABP Test Logfile

6.5 Overview of the Generic Gateway Test Environment

The generic gateway test system includes three level hierarchical structure. They are tested model, tester, and verifier. The tested model includes a generic gateway communication channels and users on each subnetworks. The tester monitors the behavior of the tested system. Finally, the verifier establishes test scenarios during test session and it analyzes the result of each tested cases.

6.5.1 Generic Gateway Test Model

The gateway test model includes 7 modules, two users, two channels, two SNPBs, and a PNB. The user modules are simple communication nodes which reside on each subnetworks. The user module is connected to the SNPB through a channel. Finally, the PNB interconnects the two communicating SNPBs (see Figure 6.10). The SNPB and PNB modules are directly converted from the LOTOS specification of the generic gateway. However, the abstract data types can not be directly converted into CLIPS, because current CLIPS does not support the functional representation. The abstract data objects must be blended into rules and facts, as concrete data objects.

While we consider the system as a finite state machine (FSM). The system's status can be identified by a state variable and the value of the state variable which represents the system's status. CLIPS is not a procedural language and the temporal ordering conditions must be represented in the predicate part of the protocol rules. The state transitions are reflected as removing the conditions which activate the current rule and creating the condition which activates the next rule.

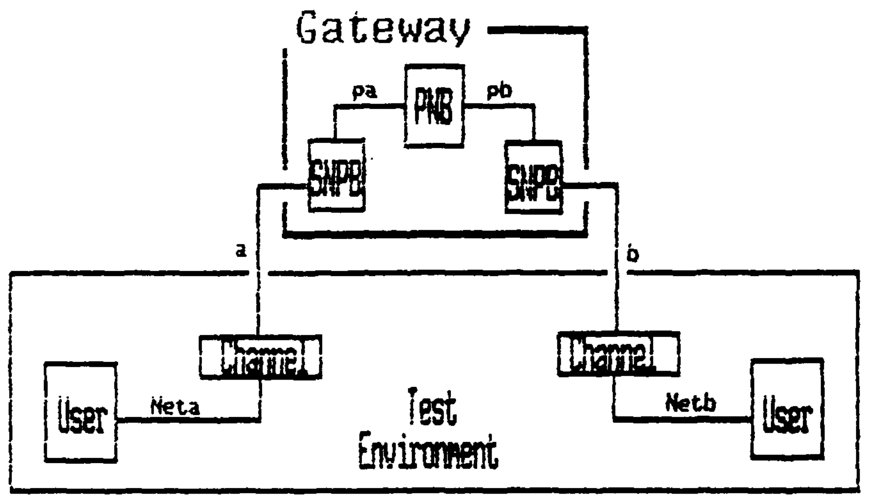


Figure 6.10 Structure of Generic Gateway Test System

The following example (Figure 6.11) shows a relationship between finite state machine, LOTOS, and CLIPS. Suppose there are two states P and Q, P and Q are to be executed sequentially. The LOTOS (middle) represents the FSM by sequential composition symbol ';' between two processes P and Q. The CLIPS (right) represents the FSM by two rules, Process_P and Process_Q. The Process_P is activated by the predicate fact (p), and the Process_Q is activated by the predicate fact (q). The temporal order of those two processes is maintained by the predicate fact (q) which is asserted by the Process_P.

Suppose the two process P and Q execute simultaneously (Figure 6.11). Then the "P;Q" in the LOTOS have to be replaced by "P || Q" (|| is a parallel operator), In the CLIPS representation, the predicate condition (p) and (q) must be initially provided to activate the Process_P and Process_Q. The result of both LOTOS and CLIPS representations will satisfies the requirements of concurrent execution.

```
FSM                    LOTOS         CLIPS

P:    goto Q  <===>    P;    <===>  (defrule Process_p
                                       (p)
Q:                     Q              =>
                                       (assert (q))
                                      )

                                     (defrule Process_Q
                                        (q)
                                      =>
                                       )
```

Figure 6.11 Conversion LOTOS processes to CLIPS rules

In LOTOS, the communication between processes is represented as a tuple of a gate and a finite set of messages. Suppose the process P sends a message 'm' to a process Q through gate g (Figure 6.12). The activity is represented as "g!m" at the message sender (process P) and as "g?n [n>0]" at the message receiver (process Q) in the LOTOS representation. Because the each rule in CLIPS has atomic property, the conversion from LOTOS to CLIPS requires special care, such as the each process of the LOTOS must be split into pre-event rule and post-event rule. In this particular example, the process P is represented by two rules which are identified as Process_P1 and Process_P2. The Process_P1 represents the pre-event activity P1 and g!m. And the Process_P2 represents the post-event activity P2. The same way the process Q is split into Process_Q1 and Process_Q2. In the CLIPS the message is represented as a fact which is created by the sender and consumed by the receiver. The message fact synchronizes between processes, sender and receiver. In this example the process Q have to wait for the message fact "(g n&:(> n 0))."

The above conversion techniques in the Generic Gateway specification in the LOTOS can be easily converted into the CLIPS representation. The states of the communicating entities (sender or receiver) are represented as a set of rules, and each rule represents the state of each entities. The rule is activated when the conditions of the rule are satisfied and the state transition is done by creating new facts which can activate the next rule which representing next state.

```
LOTOS                          CLIPS

process P:=                  (defrule Process_P1
      P1;                       (p1)
           g!m;                    =>
      P2                        P1
                                (assert (g m))
                                (assert (p2))
                               )

                              (defrule Process_P2
                                 (p2)
                               =>
                                 P2
                                )

process Q :=                    (defrule Process_Q1
      Q1;                        (q1)
      g?n [n >0];              =>
      Q2                         Q1
                                 (assert (q2))
                                )

                              (defrule Process_Q2
                                  (q2)
                                  (g n&:(>n 0))
                                =>
                                  Q2
                                 )
```

Figure 6.12 Conversion LOTOS Interprocess Communication to CLIPS rules

6.5.2 Generic Gateway Tester

The tester of the generic gateway is a set of rules which acts as an observer of the gateway's behavior during the system execution. The tester monitors various properties of the generic gateway. Which includes various protocol negotiations, buffer history, communication path establishment and termination.

6.5.3 Generic Gateway Verifier

The generic gateway verifier is responsible various tasks. First, it is responsible to manage a test environment such as individual test scenario can be established by the verifier. Secondly, it is responsible to analyze test result which produced by the tester. By the result above, the verifier generates the next test scenario and repeats the simulation on the selected scenario until all possible cases are tested.

CHAPTER 7

SUMMARY AND CONCLUSION

7.1 Summary

The generic gateway's characteristics can be summarized as follows:

a. It provides a communication protocol conversion up to transport layer;

b. The connectionless and connection-oriented subnetworks or their services can be interoperable;

c. The reliable data transport is expected by each individual subnetwork independently, because the reliable data transport control can not be expected as a global level by the their control mismatch;

d. The generic gateway is decomposed with two distinguishable modules, subnetwork independent and subnetwork dependent blocks;

e. The each subnetwork more specifically subnetwork dependents blocks communicate with subnetwork independent block, through the universal service access points;

f. Each subnetwork interfacing modules can be designed and implemented independently. [illegible]

[illegible]

The formal specification of the generic gateway is provided with formal specification language LOTOS which is proposed as a formal specification language. Appendix A contains the LOTOS specification of the generic gateway.

Finally, the generic gateway testing model is constructed and verification techniques are demonstrated with a modified constraint oriented language CLIPS. The approach used in this research shows that protocols and designs for gateways can be developed using formal specification techniques.

Eventually, the ISO LOTOS and ESTELLE languages will be developed with compilers and executable environments. These efforts should be followed for applicability to gateway protocol design.

7.2 Future Suggestion

The aggressive specification of the generic gateway has not been finished at this stage, especially the gateway specific functions. The gateway-to-gateway protocols and routing algorithms are not included and must be developed. The verifiable testing tool has a lot of room for improvement. A few suggestions for the improvement on the CLIPS are: a) use abstract data type representation; and b) use monitor routines which independently operable to rule base and inference engine.

APPENDIX A

```
Specification Generic_Gateway[a,b]:noexit

(* --------------------------------------------------------
  ===== Generic Gateway Specification with LOTOS ======

1. Global Type Definitions


  ----------------------------------------------------- *)


type Address ...    endtype     (* Address structure  *)

type Data                       (* Data structure     *)
   opns D0: -> Data             (* Empty Data         *)
   ...
endtype

type PktType is
   sorts PktType
   opns ConReq, ConInd, ConRes, ConCnf,
        DisReq, DisInd, AckReq,AckInd,
        DataReq, DataInd, ...: -> PktType
endtype

type Direction is
   sorts Direction
   opns LOCAL, REMOTE:  -> Direction
endtype

type  AddressPair is Address with
 sorts Addresspair
 opns  createpair: Address, Address -> Addresspair
       SrcAddress:  Addresspair -> Address
       Des Address: Addresspair -> Address
 eqns forall a,b: Addresspair
      ofsort Address
      SrcAddress(createpair(a, b)) = a
      DestAddress(createpair(a,b)) = b
endtype

type NetType                            (* type of subnetwork *)
```

```
  sorts NetType
  opns  Connection_Oriented: -> NetType
        Connectionless:  -> NetType
endtype

(* ------------------------------------------------------
   Each subnetwork characteristics information

   ----------------------------------------------------- *)
type BasicNetInfo is NetType, Integer, Boolean, ... with

 sorts Netinfo

 opns
     Netprot:      Netinfo -> NetProt
     Transprot:    Netinfo -> TransProt

     Blocking:     Netinfo -> Boolean
     Segmenting:   Netinfo -> Boolean

     Pktsizelimit:  Netinfo -> Integer
     Pktdelaylimit: Netinfo -> Integer
     Condelaylimit: Netinfo -> Integer

     MAXPKTSIZE:   -> Integer
     MAXPKTDELAY:  -> Integer
     MAXCONDELAY:  -> Integer
     MAXIDLETIME:  -> Integer

 eqns forall r: Netinfo
     ofsort Integer
     Pktsizelimit(r) = MAXPKTSIZE
     Pktdelaylimit(r) = MAXPKTDELAY
     Condelaylimit(r) = MAXCONDELAY

     ...
endtype

(* ------------------------------------------------------
   Formal packet structure

   ----------------------------------------------------- *)


(* *******************************************************
2. Behavior Expressions
```

The behavior specification of the gateway is decomposed with two SNPBs and a PNB. One SNPB represents the connectionless subnetwork and the other SNPB represents the

```
   connection-oriented subnetwork.

   Including two SNPB specifications doesn't implies that one
   of subnetworks is in connectionless mode and the other is
   in connection-oriented mode. But, the two distinguishable
   SNPBs are provided only as examples of the SNPBs one for
   each type of subnetworks.  By that any connected
   subnetworks can be either connectionless or connection-
   oriented subnetworks depends on the implementation.

    *************************************************** *)

behavior GEN_GATEWAY[a,b]:noexit

where

(* --------------------------------------------------------
    GEN_GATEWAY:

    In this specification, generic gateway (from now on it
    will be referred as a gateway for the convineance) is
    decomposed with three seperate submodules.

    The each modules are completely disjointed each other.
    By that they are only allowed to exchange information
    through the communication channels called gates.
    In the global specification, there are 4 gates,  Two of
    them are interface gates (gate 'a' and 'b'),

    And the other two gates, gate 'pa' and 'pb', are internal
    gates which are not visible from the outside of the
    specification.


    gates:
     a, b: communicates with SMABs
     pa, pb: internal gates which are responsible for
             communication between SNPBs and PNB.
             The activity on the gates can not observable from
             outside of the gateway specification.

    parameters: none (no initial parameters are required)

    exit:
        noexit ( it is not terminating process )


    -------------------------------------------------------- *)
process GEN_GATEWAY[a,b] :noexit :=
  hide pa, pb in
  (
      SNPBStart[a,pa]
    |[pa]|
       PNBStart[pa,pb]
    |[pb]|
       SNPBStart[pb,b]
  )
endproc  (* GEN_GATEWAY *)

(* ------------------------------------------------------

   2.1 SNPB

   Because the behavior of subnetworks are not similar each
   other, they can not be represented by one specification.
   In this specification the following two specific SNPBs
   are provided 1) connectionless subnetworks 2) connection
   oriented subnetworks.

   --------------------------------------------------- *)

(* ****************************************************

   2.1.1 SNPB specification (Connectionless subnetwork)

   In this clouse the SNPB specification of the
   connectionless subnetwork is provided.

   ************************************************** *)

(* Local Data Type definitions *)

(*  Type FIFIQueue defines the first in first out queues *)
(*  The size of queue is assumed unbounded               *)

type FIFOQueue is Data with

 sorts FQueue

 opns createque: -> FQueue
      first:  FQueue -> Data
      add:    Data, FQueue -> FQueue
      Q0:     -> FQueue
      rest:   FQueue -> Data

  eqns forall x,y:Data, q:FQueue

      ofsort FQueue
      createque = Q0
      rest(createque) = Q0
      rest(add(x,createque)) = Q0
      rest(add(x,add(y,q))) = add(x, rest(add(y,q)))
```

```
        ofsort Data
        first(createque) = D0
        first(add(x,createque)) = x
        first(add(x,add(y,q))) = first(add(y,q))
endtype


type SNetInfo is BasicNetInfo renamedby
  sortnames SNetInfo for NetInfo

  opns nettype: SNetInfo -> NetType

  eqns forall r:SNetInfo

              ofsort NetType
       nettype(r) = Connectionless
       ...
endtype


type SconInfo is Direction, String with
  sorts SConInfo
  opns
      ConInitiated: SConInfo -> Direction
      SetInitDir: Direction, SConInfo -> SConInfo
      Reason: SConInfo -> String
      SetReason: String, SConInfo -> SConInfo

  eqns forall info:ConInfo, d:Direction, str:String

      ofsort Direction
      ConInitiated(SetInitDir(d,info)) = d

      ofsort String
      Reason(SetReason(str,info)) = str
         ...
endtype


type SQueues is FIFOQueue with
  sorts Ques

  opns  createques: FQueue, FQueue -> Ques
       LocQue:  Ques -> FQueue
       RemQue:  Ques -> FQueue

  eqns forall a,b: FQueue

       ofsort FQueue
       LocQue(createques(a,b)) = a
      RemQue(createques(a,b)) = b
endtype


type SConRef is AddressPair, SQueues, SconInfo with
  sorts SconRef
  opns
     C0:         -> SConRef
     CreateCon: AddressPair, Ques, ConInfo -> SConRef
     ConAddress: SConRef -> AddressPair
     ConInfo:    SConRef -> SConInfo
     ConQues:    SConRef -> Ques

  eqns forall c:SConRef, q:Ques, info:ConInfo, a:AddressPair

     ofsort Ques
            ConQues(CreateCon(a,q,info)) = q

            ofsort SConInfo
     Info(createCon(a, q, info)) = info

     ofsort AddressPair
            ConAddress(createCon(a, q, info)) = a

endtype


type SConList is Addresspair, SConRef with

  sorts SConList

  opns createcons: -> SConList
       empty: -> SConList
       Removecon: SConRef, SConList -> SConList
       Addcon: SConRef, SConList -> SConList
       GetCon: Addresspair, SConList -> SConRef
       _IsIn_ : Addresspair, SConList -> Boolean
       _IsNotIn_ : Addresspair, SConList -> Boolean

  eqns forall r,t:SConRef, c:SConList, a:Addresspair

       ofsort SConList
       Removecon(r, createcons) = empty
       Removecon(r, Addcon(r,c)) = c
       t neq r =>
         Removecon(r, Addcon(t,c)) = Addcon(t,Removecon(r,c))
       Addcon(r, Addcon(r,c)) = Addcon (r, c)
       t neq t =>
         Addcon(r, Addcon(t,c)) = Addcon(t, Addcon(r,c))

       ofsort SConRef
```

```
          GetCon(a, createcon) = C0
          ConAddress(r) eq a =>
                       GetCon(a, Addcon(r,c)) = r
          ConAddress(r) neq a =>
                   GetCon(a, Addcon(r,c)) = GetCon(a,c)

          ofsort Bool
          a IsIn createcons = false
          a neq ConAddrress(t) =>
                    a IsIn Addcon(t,c) = a IsIn c
          a eq  ConAddress(t)  =>
                    a IsIn Addcon(t, c) = true
          a IsNotIn c  =  not(a IsIn c)

endtype (* SConList *)


type SNPktInfo is PktType ,,, with
  sorts SInfo
  opns
     createsinfo: PktType -> SInfo
     pkttype:    SInfo -> PktType
      ...
  eqns  forall p:PktType
     ofsort PktType
     pkttype(createsinfo(p)) = p
endtype

type SNPkt is PktType, Data, Addresspair, ... with
  sorts SNPkt

  opns data: SNPkt -> Data
     PktAddress:  SNPkt -> Addresspair
     createpkt:   Addresspair, SInfo, Data -> SNPkt
     pktinfo:     SNPkt -> SInfo
     ....

  eqns forall p:Pkt, a: Addresspair, i:SInfo, d:Data

     ofsort Data
     data(createpkt(a,i,d)) = d

     ofsort Addresspair
     PktAddrpair(createpkt(a,i,d)) = a

     ofsort SInfo
     pktinfo(createpkt(a,i,d)) = i

endtype (* SNPkt *)
```

```
(* ****************************************************
   2.1.2 Behavior Specification of SNPB
         (Connectionless Mode)

   ************************************************** *)

(* -----------------------------------------------------

 SNPBStart:


 SNPBStart calls two subprocesses.

 SNPBInit is responsible to initialize internal static
 information, and to send the information to the PNB.

 SNPB is a main body of the SNPB module and it is only
 valid when the SNPBInit terminates successfully

 gates:
    a: to communicates with SMAB
    b:      ""       with PNB

 parameters:
    none

 exit:
    noexit
   -----------------------------------------------------   *)

process SNPBStart[a,b] :noexit :=
   SNPBInit[b](|info, cons)
>>
   accept info:SNetInfo, cons:SConList in
   SNPB[a,b](info,cons)

where
(* ------------------------------------------------------

 SNPBInit:

     Initializes the SNPB then sends the information to the
     PNB.  Also the information of the SNPB will be passed
     to SNPBStart.

 gates:
    b: with PNB

 parameters:
    none
 exit:
    info: SNetInfo   - local subnetwork information
    cons: SConList   - Connection list
   -----------------------------------------------------   *)
process SNPBInit[b] :exit (SNetInfo, SConList):=
   i(|Mynetinfo, cons)  (* internal function, which produces
                           subnetwork infromation and initial
                           connection reference list.  *)
>>
   b ! Mynetinfo;
   exit (Mynetinfo, cons)
endproc  (* SNPBInit *)


(* ----------------------------------------------------------

 SNPB:

 The process SNPB has four subprocesses which operate
 concurrently and they are synchronized by the activities on
 the internal gates 'ga', 'gb', and 'ds'.

 The two processes, SNPBUInterface and SNPBDInterface, are
 responsible to exchanges packets with other modules (PNB,
 SMAB).

 The SNPBConHandler communicates with SNPBInterfaces and
 SNPBProtocol.  It is responsible to create new connection
 reference when it is required, and remove the connection
 reference when it is no more needed.


 Finally the process SNPBProtocol is the main body of SNPB
 which provides protocol services.

  gates:
    a: communicates with SMAB
    b:    ""        with PNB
    ga, gb, ds: internal gates

 parameters:
    info:SNetInfo,
    cons:SConList

 exit:
    noexit
 ----------------------------------------------------------  *)
process SNPB[a,b] (info:SNetInfo, cons:SConList) :noexit :=
   hide ga, gb, ds in
    (   SNPBConHandler[ds] (info, cons)
      |[ds]|
       (     SNPBDInterface[a,ga,ds] (info)
        |[ga]|
             SNPBProtocol[ga,gb,ds] (info)
        |[gb]|
             SNPBUInterface[b,gb,ds] (info)
       )
   )
endproc (* SNPB *)

(* ---------------------------------------------------
 SNPBDInterface:

        Communicates with SMAB.

 gates:
    a:  communicates with SMAB
    ga:      ''      with SNPBProtocol (internal gate)
    ds:      ''      with SNPBConHandler (internal gate)
 parameters:
    info:SNetInfo

 exit:
    none
   -------------------------------------------------  *)
  process SNPBDInterface[a,ga,ds] (info:SNetInfo): exit :=
   ( a ?pkt:SNPkt, ds !PktAddress(pkt);
     ds ?con;
     ga !pkt !con
   []
     ga ?pkt:SNPkt;
     a !pkt
   )
  >>
    SNPBDInterface[a,ga,ds] (info)  (* repeats the process *)
  endproc (* SNPBUInterface *)

(* ---------------------------------------------------

 SNPBUInterface:

   communicates with PNB.

 gates:
    b:  communicates with PNB
    gb:      ''      with SNPBProtocol (internal gate)
    ds:      ''      with SNPBConHandler (internal gate)

 parameters:
    info:SNetInfo
 exit:
    none
   ---------------------------------------------------------  *)
 process SNPBUInterface[b,gb,ds] (info:SNetInfo) :exit :=
   ( b ?pkt:PNPkt, ?con:SConRef;
     i(pkt|pkt');
     [con eq C0] => ds ! 'new, ?con;
     gb !pkt', !con
   []
     gb ?pkt:SNPkt, ?con:SConRef;
     i(pkt|pkt');
     b !pkt', !con
   )
 >>
    SNPBUInterface[a,ga,ds] (info)
endproc (* SNPBUInterface *)

(* -------------------------------------------------------
 SNPBConHandler:

  Manages connection entries.

 gates:
    a:  communicates with other processes in SNPB

 parameters:
    info:SNetInfo,
    cons:SConList

 exit:
    none
   ------------------------------------------------------  *)

  process SNPBConHandler[ds] (info:Netinfo, cons:SConList):
                                             exit :=
   ( ds ? a:Addresspair;
     [ a IsIn cons] ->
        ds !Getcon(a,cons)
   []
     [ a IsNotIn cons] ->
        let con = CreateCon(a,
                                   createques(createque,createque),
                                   info);
        let cons = Addcon(con, cons);
        ds !con
     )
   []
     (ds ?con:SconRef, ? remove:Command;
      let cons = RemoveCon(con, cons);
     )
   []
```

```
     (ds ?con:SConRef, ?new:SConRef;
      let cons = AddCon(new, RemoveCon(con, cons));
     )
>>
   SNPBConHandler[ds] (info, cons)

endporc (* SNPBConHandler *)


(* ----------------------------------------------------------
 SNPBProtocol:

   The first statement of the process is a choice expression.
  The operator choice selects one of the reference entries
   and their choice is non-deterministic. As far as the
   connection reference is selected the activity of the SNPB
   will be carried by the reference which is referred as
   'con'.

 gates:
    a: communicates with SNPBDInterface
    b: communicates with SNPBUInterface
    ds:   "          with SNPBConHandler

 parameters:
    info:SNetInfo

 exit:
    none
   -------------------------------------------------------   *)
process SNPBProtocol [a,b,ds] (info:SNetInfo):exit :=
 choice con:SConRef [] =>
   (
    (  DataIndM[a] (con | x);
       let new = CreateCon(ConAddress(con),
                 CreateQues(add(LocQue(ConQues(con)),x),
                            RemQue(ConQues(con))),
                        ConInfo(con));
       ds !con !new; exit
     []
       DataReqP[b] (con | x);
       let new = CreateCon(ConAddress(con),
                 CreateQues(LocQue(ConQues(con)),
                            add(RemQue(ConQues(con)),x),
                        ConInfo(con));
       ds !con !new; exit
     []
       [first(LocQue(ConQues(con))] ->
          DataIndP[b] (first(LocQue(ConQues(con)))
       >>
          AckReqP[b](con);
```

```
                AckReqM[a](con);
                let new = CreateCon(ConAddress(con),
                          CreateQues(rest(LocQue(ConQues(con)),x),
                                     RemQue(ConQues(con))),
                               ConInfo(con));
               ds !con !new; exit
      []
        [FirstRQue(con)] ->
           DataReqM[a] (first(Rque(con)))
         >>
           AckIndM[a](con);
           AckIndP[b](con);
           let new = CreateCon(ConAddress(con),
                     CreateQues(LocQue(ConQues(con)),
                                rest(RemQue(ConQues(con)),x),
                               ConInfo(con));
           ds !con !new; exit
        )
     [>
       [Timeout(con)] ->
          ds !con, ! 'remove; exit
    )
   ||
     SNPBProtocol[a,b,ds] (info)
  endporc (* SNPBprotocol *)

  (* -------------------------------------------------------
     SNPB Service primitives


     ---------------------------------------------------- *)
  (* Receives a Data Unit from PNB *)
  process DataReqP [c] (con:SConRef) :exit (Data) :=
        c ?pkt:SNPkt [pkttype(pkt) Is DataReq],
        ?con':SConRef [con' eq con];
      exit(Data(pkt))
  endproc

  (* Sends a Data Unit to SMAB *)
  process DataReqM [c] (con:SConRef, data:Data): exit :=
        i(con,data|pkt)          (* Build a DataReq *)
        c !pkt:SNPkt             (* Build a SNPkt *)
  endproc


  (* Receive a Data Unit from SMAB *)
  process DataIndP [c] (con:SConRef) :exit (Data)  :=
        c ?pkt:SNPkt [pkttype(pkt) Is DataInd],
        ?con':SConRef [con' eq con];
      exit(Data(pkt))
  endproc
```

```
(* Sends a Data Unit to PNB *)
process DataIndM [c] (con:SConRef, data:Data):exit =
      i(data|pkt)                 (* Build a DataInd *)
      c !pkt:SNPkt
endproc

(* receives a AckReq from PNB *)
process AckReqP [c] (con:SConRef) :exit  :=
      c ?pkt:SNPkt [pkttype(pkt) Is AckReq],
      ?con':SConRef [con' eq con];
endproc

(* Sends a AckReq to SMAB *)
process AckReqM [c] (con:SConRef):exit :=
      i(con|pkt);                 (* Build a AckReq *)
      c !pkt:SNPkt
endproc

(* receive a AckInd from SMAB *)
process AckIndM [c] (con:SConRef) :exit  :=
      c ?pkt:SNPkt [pkttype(pkt) Is AckInd],
      ?con':SConRef [con' eq con];
endproc

(* Sends a AckInd to PNB *)
process AckIndP [c] (con:SConRef):exit :=
      i(con|pkt);               (* Build a AckInd *)
      c !pkt:SNPkt
endproc

endproc (* SNPBStart *)

(* ******************************************************
   2.1.3 SNPB Specification
         (Connection-oriented Mode)


   **************************************************** *)

(* ------------------------------------------------------
   Local Data Type definitions


   ---------------------------------------------------- *)

(* Type WINQueue defines a window queue *)

type WINQueue is Data with    (* Window Queue is used for the
                                 Connection oriented
                                 Subnetwork *)
   sorts  WQueue
   opns   create:  -> WQueue
          Q0:  -> WQueue
          nth: Integer, WQueue -> Data
          add: Integer, Data, WQueue -> WQueue

   eqns   forall x y:Data, n m: Integer, q:WQueue
          ofsort Data
          nth(n, create) =  D0
          nth(n, add(n, x, q)) = x
              m neq n =>
          nth(n, add(m, x, q)) = nth(n,q)
          nth(n, rest(n,q)) = D0

          ofsort Boolean
          is_empty(D0) = true
          is_empty(add(n, x, q)) = false

          ofsort WQueue
          add(n,x,add(n,y,q)) = add(n,x,q)
              n neq m =>
          add(n,x,add(m,y,q)) = add(m,y,add(n,x,q))
          rest(n, create) = Q0
          rest(n, add(n,x,q)) = q
              n neq m =>
          rest(n, add(m,x,q)) = add(m, x, rest(n,q))

endtype

type SQueues is WINQueue with
  sorts Ques

  opns  createques: WQueue, WQueue -> Ques
        LocQue:  Ques -> WQueue
        RemQue:  Ques -> WQueue

  eqns forall a,b: WQueue

        ofsort WQueue
        LocQue(createques(a,b)) = a
        RemQue(createques(a,b)) = b
endtype

type SNetInfo is BasicNetInfo renamedby
  sortnames SNetInfo for NetInfo
  opns nettype: SNetInfo -> NetType

  eqns forall r:SNetInfo

               ofsort NetType
        nettype(r) = Connection_Oriented
```

```
     ...
endtype

type SconInfo is Direction, String, Integer with
  sorts SConInfo, SConState
  opns
      Created:       ->SConState
      Transfer:      ->SConState

      ConInitiated: SConInfo -> Direction
      SetDirection: Direction, SConInfo -> SConInfo
      ReversInit:   SConInfo -> SConInfo
      SetState:     ConState, SConInfo -> SConInfo
      SetLastSent:  SConInfo, Integer, Direction -> SConInfo
      NexttoSend:   SConInfo, Direction -> Integer
      LastSent:     SConInfo, Direction -> Integer
      Reason:       SConInfo -> String
      SetReason:    String, SConInfo -> SConInfo

 eqns forall info:ConInfo, d:Direction, str:String,
             i: Integer, s:SConState

      ofsort Direction
      ConInitiated(SetDirection(d,info)) = d

      ofsort String
      Reason(SetReason(str,info)) = str


      ofsort SConInfo
      ReversInit(SetDirection(LOCAL,info)) =
                                 SetDirection(REMOTE, info)
      ReversInit(SetDirection(REMOTE,info)) =
                                 SetDirection(LOCAL, info)

      ofsort SConState
      state(SetState(s,info)) = s

             ofsort Integer
      NexttoSend(SetLastSent(info, i, d)) = i + 1
      LastSent(SetLastSent(info, i, d), d) = i
      ...

endtype


type SConRef is
     (* same as Connectionless SNPB SConRef *)
endtype

type SConList is
```

```
     (* same as Connectionless SNPB SConList *)
endtype

type SNPktInfo is PktType, Integer, ... with
  sorts SInfo
  opns
     createsinfo: PktType, Integer -> SInfo
     pkttype:    SInfo -> PktType
     pktseq:     SInfo -> Integer
      ...
  eqns  forall p:PktType, i:Integer
     ofsort PktType
     pkttype(createsinfo(p,i)) = p
     pktseq(createsinfo(p,i)) = i

endtype

type SNPkt is
     ...
endtype


(* ***************************************************************
2.1.4 Behavior Specification of SNPB
      (Connection-oriented Mode)


   *************************************************** *)

(* ------------------------------------------------------

 SNPBStart:

 gates:
    a: communicates with SMAB
    b:       ""      with PNB

 parameters:
    none

 exit:
    noexit
   ------------------------------------------------   *)

process SNPBStart[a,b] :noexit :=
   SNPBInit[b](|info, cons)
>>
  accept info:SNetInfo, cons:SConList in
     SNPB[a,b](info,cons)

where
```

```
(* -----------------------------------------------------

 SNPBInit:

 gates:
    b: with PNB

 parameters:
    none

 exit:
    info: SNetInfo  - local subnetwork information
    cons: SConList  - Connection list
   -----------------------------------------------------  *)
process SNPBInit[b] :exit (SNetinfo, SConList) :=
   i(|Mynetinfo, cons);
    (* Initializes internal processes, generates the
         Subnetwork information, and sends it to the PNB *)
   b ! Mynetinfo;
   exit (Mynetinfo, cons)
endproc  (* SNPBInit *)

(* -----------------------------------------------------

 SNPB:

  gates:
    a: communicates with SMAB
    b:           ""  with PNB
    ga, gb, ds: internal gates

 parameters:
    info:SNetInfo, cons:SConList

 exit:
    noexit
   -----------------------------------------------------  *)


process SNPB[a,b] (info:SNetInfo, cons:SConList) :noexit :=
   hide ga, gb, ds in
    (  SNPBConHandler [ds] (info, cons)
     |[ds]|
       (   SNPBDInterface[a,ga,ds] (info)
        |[ga]|
           SNPBProtocol[ga,gb,ds] (info)
        |[gb]|
           SNPBUInterface[b,gb,ds] (info)
       )
   )
endproc (* SNPB *)
```

```
(* --------------------------------------------------------

 SNPBDInterface:


 gates:
    a:  communicates with SMAB
    ga:      ""      with SNPBProtocol (internal gate)
    ds:      ""      with SNPBConHandler (internal gaet)

 parameters:
    info:SNetInfo

 exit:
    none
   ---------------------------------------------------------  *)
  process SNPBDInterface[a,ga,ds] (info:SNetInfo) :exit :=
    (   a ?pkt:SNPkt, ds !PktAddress(pkt);
        ds ?con;
        ga !pkt !con
      []
        ga ?pkt:SNPkt; a !pkt
     )
   >>
        SNPBDInterface[a,ga,ds] (info)
  endproc (* SNPBUInterface *)

(* --------------------------------------------------------

 SNPBUInterface:


 gates:
    b:  communicates with PNB
    gb:       ''     with SNPBProtocol (internal gate)
    ds:       ''     with SNPBConHandler (internal gate)

 parameters:
    info:SNetInfo

 exit:
    none
   -------------------------------------------------------  *)
  process SNPBUInterface[b,gb,ds] (info:SNetInfo) :exit :=
    (   b ?pkt:SNPkt, ?con:SConRef;
        i(pkt|pkt');     (* Convert pkt to PNB Structure *)
        [con eq C0] => ds ! 'new, ?con;  (* new con ref. *)
        gb !pkt', !con;
      []
        gb ?pkt:SNPkt ?con:SConRef;
```

```
         i(pkt|pkt');      (* Convert pkt to PNB Structure *)
         b !pkt', !con
      )
       >>
         SNPBUInterface[a,ga,ds] (info)
   endproc (* SNPBUInterface *)

  (* -------------------------------------------------------
   SNPBConHandler:


   gates:
      a:  communicates with other processes in SNPB

   parameters:
      info:SNetInfo,
      cons:SConList

   exit:
      none
     ----------------------------------------------------   *)
  process SNPBConHandler[ds] (info:Netinfo, cons:SConList):
                                                      exit :=
  (
   ( ds ? a:Addresspair;
      [ a IsIn cons] ->
        ds !Getcon(a,cons)
     []
      [ a IsNotIn cons] ->
        let con = CreateCon(a,
                            createques(createque,createque),
                            info);
        let cons = Addcon(con, cons);
        ds !con;
   )
  []
    ds ?con:SconRef, ?remove:Command; (* Remove ConRef *)
    let cons = RemoveCon(con, cons)
  []
    ds ?con:SconRef, ?new:SConRef;      (* Update ConRef *)
    let cons = AddCon(new, RemoveCon(con, cons))
  )
  >>
     SNPBConHandler[ds] (info, cons)
  endporc (* SNPBConHandler *)


  (* -------------------------------------------------------
   SNPBProtocol:

   gates:
    a: communicates with SNPBDInterface
    b:        ""     with SNPBUInterface
    ds:       ""     with SNPBConHandler

 parameters:
    info:SNetInfo

 exit:
    none
   ----------------------------------------------   *)
process SNPBProtocol [a,b,ds] (info:SNetInfo): exit :=
choice con:Ref [] =>
  [state(Info(con)) Is created] ->
  (
         ConIndM[a](con);
         let new = CreateCon(ConAddress(con),
                             ConQues(con),
                             SetDirection(Local,Info(con)));
         ds !con, !new;
         SEND_CONIND[a,b,ds](info, new); exit
       []
         ConReqP[b](con);
         let new = CreateCon(ConAddress(con),
                             ConQues(con),
                             SetDirection(Remote,Info(con)));
         ds !con, !new;
         SEND_CONIND[a,b,ds](info, new); exit
   )
 ||
   SNPBprotocol [a,b,ds] (info)
endporc (* SNPBprotocol *)

 (* --------------------------------------------------
   SEND_CONIND:

   If the disconnection request is received from the
   connection initiating party the connection will be closed.
   And if the another connection request is received from the
   connection responder the connection reference will enter
   the double connection resolution process.

 gates:
    a: communicates with SNPBDInterface
    b:      ""      with SNPBUInterface
    ds:       ""    with SNPBConHandler

 parameters:
    info:SNetInfo,
    con: SConRef

 exit:
    none
   ---------------------------------------------------------  *)
process SEND_CONIND  [a,b,ds] (info:SNetInfo, con:SConRef):
                                                 exit :=

 [ConInitiated(Info(con)) = LOCAL] ->
  ( ConIndP[b](con);
    WAIT_CONFIRM [a,b,ds](info, con);exit

  []
    DisIndM[a] (con);
    ds !con, !'remove;exit
  []
    ConReqP[b] (con);
    DOUBLE_CONNECT[a,b,ds] (info, con);exit
  )
[]
 [ConInitiated(Info(con)) = REMOTE] ->
  ( ConReqM[a](con);
    WAIT_CONFIRM [a,b,ds](info, con);exit
  []
    DisReqP[b] (con);
    ds !con, !'remove;exit
  []
    ConIndM[a] (con);
    DOUBLE_CONNECT[a,b,ds] (info,con);exit
    )

endproc (* SEND_CONIND *)

(* -----------------------------------------------------------
   DOUBLE_CONNECT:

   In the generic gateway only one connection is allowed for
   each pair of communication entities. By that only one
   direction of the connections will be granted and the other
   must be rejected (Connection request collision).

 gates:
    a: communicates with SNPBDInterface
    b:      ""      with SNPBUInterface
    ds:      ""     with SNPBConHandler

 parameters:
    info:SNetInfo,
    con: SConRef

 exit:
    none
   ---------------------------------------------------------  *)
process DOUBLE_CONNECT[a,b,ds] (info:SNetInfo, con:SConRef):
```

```
                                                exit :=
 [ConInitiated(Info(con)) = LOCAL] ->
   ( DisIndM[a] (con);
     SEND_CONIND[a,b,ds] (info,
                                CreateCon(ConAddress(con),
                                        ConQues(con),

ReversInit(Info(con))));
     exit
   []
     DisReqP[b] (con);
     SENDCONIND[a,b,ds] (info,con); exit
   []
   let reason = "Connection already in progress from remote";
     let con' = CreateCon(ConAddress(con),
                                ConQues(con),
                                SetReason(reason,Info(con)));
     DisReqM[a](con');
     SEND_CONIND[a,b,ds] (info,
                                CreateCon(ConAddress(con'),
                                        ConQues(con'),
                                       ReversInit(Info(con'))));
     exit
    )
[]
 [ConInitiated(Info(con)) = REMOTE] ->
   ( DisIndM[a] (con);
     SENDCONIND[a,b,ds] (info,con);exit
   []
     DisReqP[b] (con);
     SEND_CONIND[a,b,ds] (info,
                                CreateCon(ConAddress(con),
                                        ConQues(con),
                                        ReversInit(Info(con))));
     exit

   []
   let reason = "Connection already in progress from remote";
     let con' = CreateCon(ConAddress(con),
                                ConQues(con),

SetReason(reason,Info(con)));
     DisReqM[a](con');
     SENDCONIND[a,b,ds] (info, con');exit
    )

endproc.  (* DOUBLE_CONNECT *)

(* ---------------------------------------------------------
   WAIT_CONFIRM:
```

```
   After the connection indication is sent to the connection
   responder, the gateway must waits for the connection
   response from the connection responder.


 gates:
   a: communicates with SNPBDInterface
   b:        ""    with SNPBUInterface
   ds:      ""     with SNPBConHandler

 parameters:
   info:SNetInfo,
   con: SConRef

 exit:
   none
   ---------------------------------------------------  *)
process WAIT_CONFIRM [a,b,ds] (info:SNetInfo, con:SConRef):
                                                  exit :=
 [ConInitiated(Info(con)) = LOCAL] ->
   (  ConResP[b] (con);
      CON_PROC[a,b,ds] (info, con);exit
    []
      DisReqP[b] (con);
      DisReqM[a] (con);
      ds !con, !'remove;exit
    []
      DisIndM[a] (con);
      DisIndP[b] (con);
      ds !con, !'remove;exit
    )
[]
 [ConInitiated(Info(con)) = REMOTE] ->
   (  ConCnfM[a] (con);
      CON_PROC[a,b,ds] (info, con);exit
    []
      DisReqP[b] (con);
      DisReqM[a] (con);
      ds !con, !'remove;exit
    []
      DisIndM[a] (con);
      DisIndP[b] (con);
      ds !con, !'remove;exit
    )
endproc.

(* -------------------------------------------------------
   CON_PROC:

   This is the final stage of connection establishment phase.
   During this phase the connection conform is sent to the
   connection requester.

 gates:
    a: communicates with SNPBDInterface
    b:         ""    with SNPBUInterface
    ds:       ""     with SNPBConHandler

 parameters:
    info:SNetInfo,
    con: SConRef

 exit:
    none
   ---------------------------------------------------  *)
process CON_PROC [a,b,ds] (info:SNetInfo, con:SConRef):
                                                   exit :=
 [ConInitiated(Info(con)) = LOCAL] ->
   DataReqP[b] (con |x)·
   add_data(con, x, REMOTE |con');
   AckIndP[b] (con', Ack(Info(x)));
   CON_PROC [a,b,ds](info, con');exit
[]
 [ConInitiated(Info(con)) = REMOTE] ->
   DataIndM[a] (con |x);
   add_data(con, x, LOCAL |con');
   AckReqM[a] (con, Ack(Info(x)));
   CON_PROC [a,b,ds](info,con');exit
[]
   DisReqP[b] (con);
   DisReqM[a] (con);
   ds !con, !'remove;exit
[]
   DisIndM[a] (con);
   DisIndP[b] (con);
   ds !con, !'remove;exit
[]
 (
   [ConInitiated(Info(con)) = LOCAL] ->  ConCnfP[b](con)
 []
   [ConInitiated(Info(con)) = REMOTE] -> ConResM[a](con)
 DATA_TRANSFER[a,b,ds] (ConAddress(con),
                        CreateCon(ConAddress(con),
                                  ConQues(con),
                        SetState(Transfer,Info(con))));
  exit
 )
endproc. (* CON_PROC *)

(* --------------------------------------------------------
   DATA_TRANSFER:
```

```
 gates:
    a: communicates with SNPBDInterface
    b:      ""       with SNPBUInterface
    ds:      ""      with SNPBConHandler

 parameters:
    info:SNetInfo,
    con: SConRef

 exit:
    none
   ---------------------------------------------------  *)
process DATA_TRANSFER [a,b,ds] (info:SNetInfo, con:SConRef):
                                                 exit :=
[state(Info(con)) = Transfer] ->
 (   DataIndM[a] (con |x);
     add_data(con, x, LOCAL |con');
     DATA_TRANSFER [a,b,ds] (info, con');exit
  []
     DataReqP[b] (con |x);
     add_data(con, x, REMOTE |con');
     AckIndP[b] (con, Ack(x));
     DATA_TRANSFER [a,b,ds] (info, con);exit
  []
     [not(is_empty(LocQue(ConQues(con))))] ->
         SEND_DATA[b] (info, con, LOCAL |con');
         DATA_TRANSFER [a,b,ds] (info, con);exit
  []
     [not(is_empty(RemQue(ConQues(con))))] ->
         SEND_DATA[a] (info, con, REMOTE |con');
         DATA_TRANSFER [a,b,ds] (info, con);exit
  []
     AckIndM[a] (con |z);
     remove_data[ds](con, z, REMOTE |con');
     DATA_TRANSFER [a,b,ds] (info, con');exit
  []
     DisIndM[a] (con);
     TERMINATION[a,b,ds] (info,
                         CreateCon(ConAddress(con),
                                      ConQues(con),
                         SetState(Terminate, Info(con))),
                         LOCAL); exit
  []
     DisReqP[b] (con);
     TERMINATION[a,b.ds] (info,
                         CreateCon(ConAddress(con),
                                      ConQues(con),
                         SetState(Terminate, Info(con))),
                         REMOTE); exit
  )
 endproc. (* DATA_TRANSFER *)

(* -----------------------------------------------------
  TERMINATION:

  To provide graceful disconnection any non delivered data
  packets must be properly delivered before it releases the
  connection.

 gates:
    a: communicates with SNPBDInterface
    b:       ""      with SNPBUInterface
    ds:       ""     with SNPBConHandler

 parameters:
    info:SNetInfo,
    con: SConRef

 exit:
    none
   ------------------------------------------------------  *)
process TERMINATION [a,b,ds] (info:SNetInfo, con:SConRef,
                                   d:Direction): exit  :=
  [d = LOCAL] ->
   (
    [not(is_empty(LocQue(ConQues(con))))] ->
        SEND_DATA[b] (info, con, LOCAL);
        TERMINATION [a,b,ds] (inf, con, LOCAL)
    >>
       DisIndP[b] (con);
       ds !con, !'remove;exit
   )
[]
  [d = REMOTE] ->
  (
    [not(is_empty(RemQue(ConQues(con))))] ->
        SEND_DATA[a] (info, con, REMOTE);
        TERMINATION [a,b,ds] (info, con', REMOTE)
    >>
       DisReqM[a] (con);
       ds !con, !'remove;exit
   )
endproc. (* TERMINATION *)

(* -----------------------------------------------------
   SEND_DATA:

 gates:
    a:  communicates with SNPBInterfece
    ds:   ""              SNPBConHandler
 parameters:
    info:SNetInfo,
    con: SConRef,
    d:Direction

 exit:
    SConRef

   ---------------------------------------------------  *)
process SEND_DATA [a] (inf:NetInfo, con:SConRef d:Direction):
                                                exit(SConRef) :=
[d = LOCAL] ->
    let seq = NexttoSend(Info(con), LOCAL);
      [ Find_data(seq, RemQue(ConQues(con))) is true ] ->
           let data = nth(seq,RemQue(ConQues(con)));
           DataIndP[a](con, data);
           let new = CreateCon(ConAddress(con),
                               ConQues(con),
                               SetLastSent(Info(con),
                                           seq,LOCAL));
           exit (new)

[d = REMOTE] ->
    let seq = NexttoSend(Info(con), REMOTE);
      [ Find_data(seq, LocQue(ConQues(con))) is true ] ->
           let data = Get_data(seq,LocQue(ConQues(con)));
           DataReqM[a](con, data);
           AckIndM[a] (con' |z);
           let new = CreateCon(ConAddress(con),
                               ConQues(con),
                               SetLastSent(Info(con),
                                           seq,REMOTE));
           exit(new)
endproc.

(* -----------------------------------------------------
   add_data:
     Inserts incoming data on queue.

 parameters:
        con:SConRef,
        data:Data,
        d:Direction,

 gates:
    none

 exit:
    SConRef - updated connection reference entry

   -----------------------------------------------------  *)
```

```
process add_data (con:SConRef, pkt:SNPkt, d:Direction):
                    exit(SConRef) :=
 [d = LOCAL] ->
    let pktseq = Getseq(Info(pkt));
    let seq = LastSent(Info(con), LOCAL);
    [pktseq > seq and pktseq < seq + Quesize(Info(con))] ->
        let new = CreateCon(ConAddress(con),
                            Createques(add(pktseq,Data(pkt),
                                           LocQue(ConQues(con))),
                                          RemQue(ConQues(con))),
                            Info(con));
       exit(new)
[]
 [d = REMOTE] ->
    let pktseq = Getseq(Info(pkt));
    let seq = LastSent(Info(con), REMOTE);
    [ pktseq > seq and pktseq < seq + Quesize(Info(con))] ->
        let new = CreateCon(ConAddress(con),
                            Createques(LocQue(ConQues(con))),
                                       add(pktseq,Data(pkt),
                                       RemQue(ConQues(con))),
                            Info(con));
        exit(new)
endproc.


(* ------------------------------------------------------------
   remove_data:
    If the last transmission was successful then the
    sequence number must be updated and also the data
    element may be removed from the queue.

    parameters:
         con:SConRef,
         x:data,
         d:Direction

    gate:
        ds:
    exit:
       SConRef
   ---------------------------------------------------------  *)
process remove_data[ds] (con:SConRef, x:data, d:Direction):
                                                exit(SConRef):=
  [GetSeq(Info(x)) = Succ(LastSent(Info(con),d))] ->
     let new = CreateCon(ConAddress(con),
                         ConQues(con),
                         IncAck(Info(con),d));
     exit(new)
endproc.
```

```
(* ********************************************************** *)

(* SNPB service primitives *)
(* ---------------------------------------------------- *)

(* Receives a ConReq from PNB *)
process ConReqP [c] (con:SConRef): exit:=
      c ?pkt:SNPkt [pkttype(pkt) Is ConReq],
      ?con':SConRef [con' eq con];
    exit
endproc

(* Sends a ConReq to SMAB *)
process ConReqM [c] (con:SConRef): exit :=
      i(con|pkt)          (* Build a ConReq *)
      c !pkt:SNPkt             (* Build a SNPkt *)
endproc


(* Receive a ConInd from SMAB *)
process ConIndP [c] (con:SConRef) :exit :=
      c ?pkt:SNPkt [pkttype(pkt) Is ConInd],
      ?con':SConRef [con' eq con];
    exit
endproc

(* Sends a ConInd to PNB *)
process ConIndM [c] (con:SConRef):exit =
      i(pkt)              (* Build a ConInd *)
      c !pkt:SNPkt
endproc

(* receives a DisReq from PNB *)
process DisReqP [c] (con:SConRef) :exit  :=
      c ?pkt:SNPkt [pkttype(pkt) Is DisReq],
      ?con':SConRef [con' eq con];
endproc

(* Sends a DisReq to SMAB *)
process DisReqM [c] (con:SConRef):exit :=
      i(con|pkt);              (* Build a DisReq *)
      c !pkt:SNPkt
endproc

(* receive a ConRes from PNB *)
process ConResP [c] (con:SConRef) :exit  :=
      c ?pkt:SNPkt [pkttype(pkt) Is ConRes],
      ?con':SConRef [con' eq con];
endproc
```

```
(* Sends a ConRes to SMAB *)
process ConResM [c] (con:SConRef):exit :=
      i(con|pkt);            (* Build a ConRes *)
      c !pkt:SNPkt
endproc

(* receive a ConCnf from SMAB *)
process ConCnfM [c] (con:SConRef) :exit  :=
      c ?pkt:SNPkt [pkttype(pkt) Is ConCnf],
      ?con':SConRef [con' eq con];
endproc

(* Sends a ConCnf to PNB *)
process ConCnfP [c] (con:SConRef):exit :=
      i(con|pkt);            (* Build a ConCnf *)
      c !pkt:SNPkt
endproc

(* ------------------------------------- *)
(* Receives a Data Unit from PNB *)
process DataReqP [c] (con:SConRef) :exit (Data) :=
      c ?pkt:SNPkt [pkttype(pkt) Is DataReq],
      ?con':SConRef [con' eq con];
    exit(Data(pkt))
endproc

(* Sends a Data Unit to SMAB *)
process DataReqM [c] (con:SConRef, data:Data): exit :=
      i(con,data|pkt)          (* Build a DataReq *)
      c !pkt:SNPkt             (* Build a SNPkt *)
endproc


(* Receive a Data Unit from SMAB *)
process DataIndP [c] (con:SConRef) :exit (Data)  :=
      c ?pkt:SNPkt [pkttype(pkt) Is DataInd],
      ?con':SConRef [con' eq con];
    exit(Data(pkt))
endproc

(* Sends a Data Unit to PNB *)
process DataIndM [c] (con:SConRef, data:Data):exit =
      i(data|pkt)              (* Build a DataInd *)
      c !pkt:SNPkt
endproc

(* receives a AckReq from PNB *)
process AckReqP [c] (con:SConRef) :exit  :=
      c ?pkt:SNPkt [pkttype(pkt) Is AckReq],
      ?con':SConRef [con' eq con];
endproc
```

```
(* Sends a AckReq to SMAB *)
process AckReqM [c] (con:SConRef):exit :=
      i(con|pkt);               (* Build a AckReq *)
      c !pkt:SNPkt
endproc

(* receive a AckInd from SMAB *)
process AckIndM [c] (con:SConRef) :exit  :=
      c ?pkt:SNPkt [pkttype(pkt) Is AckInd],
      ?con':SConRef [con' eq con];
endproc

(* Sends a AckInd to PNB *)
process AckIndP [c] (con:SConRef):exit :=
      i(con|pkt);             (* Build a AckInd *)
      c !pkt:SNPkt
endproc

endproc (* SNPBStart *)

(* ***********************************************************
  2.2 PNB specification
```

The protocol negotiation can be subclassified as either static or dynamic negotiation. The static negotiation is fundamental negotiation which is processed during the gateway initialization step. It can not be altered during the communication of individual communication session. By that the static negotiation will be performed with the each subnetwork's information which are exported from each SNPBs during gateway initialization step.

If the static negotiation terminates successfully then the two subnetworks can communicates without any degraded functionality. If the minor fixes are required then the two subnetworks will be interoperable with minor functional limitation during its operation. However, if the result is failure then the gateway will be halt and no further operations are possible.

The second class of negotiation, which is identified as a dynamic negotiation, is a connection level negotiation. Which would be applied by the connection entities.

The module structure is quite similar to the SNPB in various aspects.
First of all, the PNB is decomposed with the four

```
     separate blocks: two blocks which are responsible to
     communicate with SNPBs on each side; one which maintains
     the connection reference information; and one which is
     responsible for the protocol negotiation and service
     interface between those two subnetworks.

    ***************************************************** *)

(* ------------------------------------------------------------
   2.2.1 Type definitions


   ------------------------------------------------------- *)
type PConRef is Data, SConRef with
   sorts PConRef, Side, BufState

   opns P0:         -> PConRef
        A:          -> Side
        B:          -> Side
        Full:       -> BufState
        Empty:      -> BufState
        CreatePcon: SConRef, SConRef -> PConRef
        Con:        PConRef,Side   -> SConRef
        Con:        PConRef,Side   -> SConRef
        Put:        PConRef,Data, Side  -> PConRef
        Get:        PConRef,Side   -> Data


   eqns forall c:PConRef, s1, s2:SConRef, d:Data, x:Side

        ofsort SconRef
        Con(CreatePCon(s1, s2), A) = s1
        Con(CreatePCon(s1, s2), B) = s2
        StateBuf(Put(c,d,x), x) = Full
        StateBuf(Get(c,x),x)  = Empty
        Get(Put(c,d,x),x) = d
        Get(CreatePCon(s1,s2),x) = D0
        Get(Get(c,x),x) = D0

endtype

type PConList is  PConRef with

  sort PConList

  opns
       PL0:      -> PConRef
       GetCon: SConRef, PConList  -> PConRef
       Addcon: PConRef, PConList -> PConList
       Restcons: PConRef, PConList -> PConList
       _IsIn_ : PConRef, PConList -> Boolean
```

```
      _IsNotIn_ : PConRef, PConList -> Boolean

  eqns forall p r t:PConRef, s sl s2:SConRef, c:PConList
      ofsort PConList
      Restcon(r, PL0) = P0
      Restcon(r,Addcon(r,c)) = cons
            t neq p =>
            Restcon(r,Addcon(p,c)) = Addcon(p, Restcon(r,c))
      Addcon(r, Addcon(t, c)) = Addcon(t, Addcon(r,c))
              r eq t =>
              Addcon(r, Addcon(t, c)) = Addcon (r, c)

      ofsort PConRef
      GetCon(sl, createcons) = P0
      GetCon(sl, Addcon(CreatePCon(sl, s2), cons)) =
                                             CreatePCon(sl,s2)
      GetCon(s2, Addcon(CreatePCon(sl, s2), cons)) =
                                             CreatePCon(sl,s2)
              s neq sl and s neq s2 =>
          GetCon(s, Addcon(CreatePCon(sl, s2), cons)) =
                                              GetCon(s, cons)

      ofsort Bool
      sl IsIn createcons = false
      sl IsIn Addcon(CreatePCon(sl, s2),c) = true
      s2 IsIn Addcon(CreatePCon(sl, s2),c) = true
      s neq sl and s neq s2 =>
           s IsIn Addcon(CreatePCon(sl, s2), c)) = r IsIn c
      s IsIn c  =  not(r IsNotIn c)

endtype (* ConList *)


(* ---------------------------------------------------------
   2.2.1 Behavior specification  of PNB


   ----------------------------------------------------- *)

(* --------------------------------------------------------
   PNBStart:

   PNBStart is a start up routine which is responsible to
   initialize its internal structure.

   When the first process PNBInit terminates successfully the
  PNB can continue the operation.  However if the PNBInit
   fails to communicate or negotiate with SPNBs then the PNB
   will not be activated.

 gates:
      pa, pb: communicates with SNPB modules

 parameters:
      none

 exit:
      noexit

   ------------------------------------------------------ *)
process PNBStart[pa,pb] :noexit :=
   PNBInit[pa,pb](|neta, netb, cons)
>>
   accept neta:NetInfo, netb:NetInfo, cons:PConList in
     PNB[pa,pb](neta, netb, cons)

where
(* ----------------------------------------------------------
   PNBInit:
    This process performs two major functions: first
    it will receives the subnetwork information from SNPBs;
    second it will perform static negotiation.

    The internal functions of the static negotiation,
    'Interoperable()', are not defined at this moment.


 gates:
      pa, pb: communicates with SNPB modules

 parameters:
      none

 exit:
      neta, netb: Subnetwork Information
      cons:   Connection reference list

   ------------------------------------------------------ *)
process PNBInit[pa,pb]: exit(SNetInfo, SNetInfo, PConList):=
   pa ?neta:SNetInfo, pb ?netb:SNetInfo;
   i(|cons)  (* Create connection reference list *)

>>
    [Interoperable(neta, netb) = false] -> stop
  []
    [Interoperable(neta, netb)] ->
        exit(neta, netb, cons)
endproc

(* ----------------------------------------------------------
   PNB:
```

```
 gates:
    pa,pb:  communicates with SNPB
    ga,gb,ds:  internal gates

 parameters:
    infa:SNetInfo,
    infb:SNetInfo,
    cons:SConList

 exit:
    noexit
   -------------------------------------------------------- *)
process PNB[pa,pb](infa, infb:SNetInfo, cons:PConList):
                                                  noexit :=
   hide ga, gb,ds in
    ( PNBConHandler [ds] (info, cons)
     |[ds]|
      (
          PNBInterface[pa,ga,ds] (infa, A)
       |[ga]|
          PNBProtocol[ga,gb,ds] (infa, infb)
       |[gb]|
           PNBInterface[pb,gb,ds] (infb, B)
endproc (* PNB *)

(* -------------------------------------------------------
 PNBInterfaces:

 gates:
    a:  communicates with SNPB
    b:      ""       with PNBProtocol (internal gate)
    ds:     ""       with PNBConHandler (internal gaet)

 parameters:
    info:NetInfo

 exit:
    noexit
   -------------------------------------------------------- *)
process PNBInterface[a,b,ds] (info:SNetInfo, S:Side)
                                                 :exit :=
  (
    ( a ?pkt:PNPkt, ?acon:SConRef, ds !acon, !S;
      ds ?pcon;
      b !pkt, !pcon;
      exit
    )
  []
      b ?pkt:PNPkt, ?pcon:PConRef;
      [S = A] ->
           a !pkt, !Con(pcon, A);
       [S = B] ->
            a !pkt, !Con(pcon, B);
       exit
   )
>>
   PNBInterface[a,b,ds](info, S)
endporc (* PNBInterface *)

(* ----------------------------------------------------------
   PNBConHandler:

 gates:
    ds: communicates with other processes in PNB
        (internal gate)

 parameters:
    infa:SNetInfo,
    infb:SNetInfo,
    cons:PConList

 exit:
    noexit
   ---------------------------------------------------------- *)
process PNBConHandler[ds](infa, infb:SNetinfo,
                                cons:PConList):exit :=
  ( ds ? con:PConRef, S:Side;
    [ con IsIn cons] ->
          ds !Getcon(con,cons);
  []
     [ con IsNotIn cons] ->
          [Side = A] ->
               let pcon = createcon(acon, S0)
         []
           [Side = B] ->
               let pcon = createcon(S0, con);
        let cons = Addcon(pcon, cons);
        ds !pcon;
     )
   []
     (ds ?pcon:PconRef, ?remove:Command;  (* Remove ConRef *)
      let cons = RestCon(pcon, cons);
     )
  []
     (ds ?pcon:PconRef, ?new:PConRef;     (* Update ConRef *)
      let cons = AddCon(new, RestCon(pcon, cons));
     )
>>

 PNBConHandler[ds] (infa, infb, cons)

endporc (* PNBConHandler *)
```

```
(* --------------------------------------------------------
   PNBProtocol:

 gates:
    a:  communicates with SNPB A
    b:         ""     with SNPB B
    ds: communicates with PNBConHandler
        (internal gaet)

 parameters:
    infa:SNetInfo,
    infb:SNetInfo

 exit:
    noexit
   ---------------------------------------------------- *)
(* Main body of PNB *)
PNBProtocol [a,b,ds] (infa, infb: SNetInfo) exit :=

choice con:PConRef [] =>

 (
   (* REQUEST FROM SIDE A *)
   ( [nettype(infa) = connection_oriented] ->
      ( ConReq[a](con);
        Initial_Negotation(infa, infb, con, A |result, con');
        [result = success] =>
           ([nettype(infb) = connection_oriented] ->
                ConInd[b](con');
                WAIT_CONFIRM[a,b,ds](infa, infb, con',A);exit
           []
             [nettype(infb) = connection_less] ->
                DataTransfer[a,b,ds](infa, infb, con')
           )
        [result = fail] =>
             DisInd[a](con');
             ds !con', !'remove;exit
        )
     []
      [nettype(infa) = connection_less] ->
        DataReq[a](con |pkt);
        Put(con, Data(pkt), A);
        Initial_Negotation(infa, infb, con, A|result, con');
        [result = success] =>
           ([nettype(infb) = connection_oriented] ->
               ConRes[b](con');
               WAIT_CONFIRM [a,b,ds](infa, infb, con',A);
               exit
           []
            [nettype(infb) = connection_less] ->
```

```
                DataTransfer [a,b,ds](infa, infb, con');exit
              )
           [result = fail] =>
               ds !con',!remove;exit
[]
   (* REQUEST FROM SIDE B *)
  (  [nettype(infb) = connection_oriented] ->
     (  ConReq[b](con | pkt);
        Initial_Negotation(infb, infa, con, B|result, con');
        [result = success] =>
           ([nettype(infa) = connection_oriented] ->
               ConInd[a](con');
               WAIT_CONFIRM [b,a](infb, infa, con',B);exit
            []
             [nettype(infa) = connection_less] ->
               DataTransfer[a,b,ds](infa, infb, con');exit
             )
        [result = fail] =>
             DisInd[a](con');
             ds !con',!remove;exit
     )
     [nettype(infb] = connection_less] ->
        DataReq[b](con |pkt );
        Put(con, Data(pkt), B);
        Initial_Negotation(infa, infb, con, B|result, con');
        [result = success] =>
           ([nettype(infa) = connection_oriented] ->
               ConRes[a](con');
               WAIT_CONFIRM [a,b,ds](infb, infa, con');exit
             []
              [nettype(infa) = connection_less] ->
               DataTransfer[a,b,ds](infa, infb, con');exit
             )
        [result = fail] =>
             ds !con',!'remove;exit
     )
  )
||
   PNBProtocol[a,b,ds] (infa,infb)

endproc (* PNBProtocol *)

(* ----------------------------------------------------
   WAIT_CONFIRM:

 gates:
    a:  communicates with SNPB A
    b:            ""    with SNPB B
    ds: communicates with PNBConHandler

                                    (internal gate)
 parameters:
    infa:SNetInfo,
    infb:SNetInfo,
    cons:PConList

 exit:
    noexit

   ---------------------------------------------------- *)
process WAIT_CONFIRM [a,b,ds] (infa, infb: SNetInfo,
                               con:PConRef, S:Side):=

   (  ConRes[b] (con |con');
      Second_Negotiation(infa, infb, con'|result, con);
      (
          [result = success] ->
             ConCnf[a] (con);
             DataTransfer[a,b,ds](infa, infb, con');
        []
          [result = fail] ->
            (
             [nettype(infa) = connection_oriented] ->
                 DisInd[a](con);
                 ds !con, !'remove;exit
             [nettype(infa) = connection_less] ->
                 ds !con, !'remove;exit
            )
      )
  []
   (  ConRes[a] (con | con');
      Second_Negotiation(infb, infa, con'|result, con);
      (
          [result = success] ->
             ConCnf[b] (con);
             DataTransfer[a,b,ds](infa, infb, con);
        []
          [result = fail] ->
            (
             [nettype(infb) = connection_oriented] ->
                  DisInd[b](con);
                  ds !con, !'remove;exit
             [nettype(infb) = connection_less] ->
                  ds !con, !'remove;exit
            )
      )

endproc. (* WAIT_CONFIRM *)

(* ----------------------------------------------------
   DATA_TRANSFER:
```

```
gates:
   a:  communicates with SNPB A
   b:        ""      with SNPB B
   ds: communicates with other processes in PNB
                                          (internal gaet)

parameters:
   infa:SNetInfo,
   infb:SNetInfo,
   con:PConRef

exit:
   none

   ----------------------------------------------------- *)
process DataTransfer [a,b,ds] (infa infb:SNetInfo,
                              con:PConRef): exit :=
    [BufState(con,A) = Empty] ->
         DataReq[a](con|pkt);
         Put(con, Data(pkt), A);
         AckInd[a](con)
  []
    [BufState(con,B) = Empty] ->
         DataReq[b](con|pkt);
         Put(con, Data(pkt), B);
  []
    [BufState(con,A) = Full] ->
         DataInd[b](con, Get(con,A));
  []
    [BufState(con,B) = Full] ->
         DataInd[a](con, Get(con,B));
  []
        DisReq[a] (con);
        [BufState(con,A) = Full] ->
               DataInd[b](con, Get(con,A));
        [nettype(infb) = connection_oriented] ->
               DisInd[b] (con);
        ds !con, !'remove;exit
  []
        DisReq[b] (con);
        [BufState(con,B) = Full] ->
               DataInd[a](con, Get(con,B));
        [nettype(infa) = connection_oriented] ->
               DisInd[a] (con);
        ds !con, !'remove;exit
endproc.


(* ------------------------------------------------------ *)
process Initial_Negotation(infa, infb:NetInfo, con:PConRef,
                              S:Side): exit(Bool,ConRef) :=
    i:(infa, infb, con,S|result,con');
    exit(result, con')

endproc

process Second_Negotiation(infa, infb:NetInfo, con:PConRef
                              S:Side): exit(Bool,ConRef) :=

    i:(infa, infb, con,S|result,con');
    exit(result, con')

endproc

(* ---------------------------------------------- *)
(* PNB service primitives *)

process ConReq [c] (con:PConRef) exit(PNPkt, PConReq):=
    c ?pkt:PNPkt [pkttype(pkt) Is ConReq], ?con:PConRef;
    exit(pkt, con)
endproc

process ConInd [c] (con:PConRef) :=
   let x = GetConInd(con);
   c!x !con; exit
endproc

process ConRes [c] (con:PConRef) exit(PConRef):=
    c ?pkt:PNPkt [pkttype(pkt) Is ConRes], ?con:PConRef;
    exit(con)
endproc

process ConCnf [c] (con:ConRef) :=
    let x = GetConCnf(con);
    c!x !con; exit
endproc

process DisReq [c] (con:PConRef) :=
    c ?pkt:PNPkt [pkttype(pkt) Is DisReq] ?con:ConRef; exit
endproc

process DisInd [c] (con:PConRef) :=
   let x = GetDisInd(con);
   c!x !con;exit
endproc

(* ------------------------------------------------

     This group of processes provides a data tranfer related
     PNB service primitives.
   ------------------------------------------------------------- *)
process DataReq[c](con:PConRef): exit(PNPkt, PConRef):=
    a ?pkt:PNPkt [pkttype(pkt) Is DataReq], ?con:PConRef;
    b !pkt, ! BCon(con);exit
endproc

process AckReq[c](con:PConRef): exit(PNPkt, PConRef) :=
    a ?pkt:PNPkt [pkttype(pkt) Is AckReq], ?con:PConRef;
    b !pkt, ! BCon(con);exit
endproc

process DataInd[c](con:PConRef, pkt:PNPkt) :=
    c !pkt:PNPkt; exit
endproc

process AckInd[c](con:PConRef, pkt:PNPkt) :=
    c !pkt:PNPkt; exit
endproc

endproc (* PNB Start *)

endspec. (* Generic Gateway *)
```